SMT-RAT

1 SMT-RAT 1

2 Installation 1

2.1 Requirements . . . 1
2.2 Download . . . 2
2.3 Configuration . . . 2
2.4 Build . . . 2
2.5 Check build . . . 2

3 System architecture 2

3.1 Different libraries . . . 2
3.2 Software design . . . 3
3.3 Modules . . . 4
3.4 Strategy . . . 4
3.5 Manager . . . 4
3.6 Procedures implemented as modules . . . 4
3.7 Infeasible subsets and lemmas . . . 4

4 Modules 5

4.1 Main members of a module . . . 5
4.2 Interfaces to implement . . . 5
4.2.1 Informing about a constraint . . . 5
4.2.2 Adding a received formula . . . 6
4.2.3 Removing a received formula . . . 6
4.2.4 Checking for satisfiability . . . 6
4.2.5 Updating the model/satisfying assignment . . . 6
4.3 Running backend modules . . . 7
4.4 Auxiliary functions . . . 8
4.5 Existing module implementation . . . 8
4.6 Available modules . . . 8
4.6.1 BEModule . . . 9
4.6.2 BVModule . . . 9
4.6.3 CNFerModule . . . 9
4.6.4 CSplitModule . . . 9
4.6.5 CoCoAGBModule . . . 9
4.6.6 CubeLIAModule . . . 9
4.6.7 CurryModule . . . 9
4.6.8 EMModule . . . 9
4.6.9 ESModule . . . 9
4.6.10 FPPModule . . . 9
4.6.11 FouMoModule . . . 9
4.6.12 GBModule . . . 9
4.6.13 GBPPModule . . . 10

4.6.14 ICEModule . . . 10
4.6.15 ICPModule . . . 10
4.6.16 IncWidthModule . . . 10
4.6.17 IntBlastModule . . . 10
4.6.18 IntEqModule . . . 10
4.6.19 LRAModule . . . 10
4.6.20 LVEModule . . . 11
4.6.21 MCBModule . . . 11
4.6.22 NRAILModule . . . 11
4.6.23 NewCADModule . . . 11
4.6.24 PBGaussModule . . . 11
4.6.25 PBPPModule . . . 11
4.6.26 PFEModule . . . 11
4.6.27 SATModule . . . 12
4.6.28 STropModule . . . 12
4.6.29 SplitSOSModule . . . 12
4.6.30 SymmetryModule . . . 12
4.6.31 UFCegarModule . . . 12
4.6.32 UnionFindModule . . . 12
4.6.33 VSModule . . . 12

5 Strategies **13**

6 Using SMT-RAT **14**
6.1 Standalone solver . . . 14
6.1.1 Formula analysis . . . 15
6.1.2 Preprocessing . . . 15
6.1.3 Quantifier elimination . . . 15
6.1.4 DIMACS solving . . . 15
6.1.5 Pseudo-Boolean solving . . . 15
6.1.6 Optimization . . . 16
6.2 Embedding in other software . . . 16
6.2.1 Interface . . . 16
6.2.2 Syntax of formulas . . . 17
6.2.3 Boolean combinations of constraints and Boolean variables . . . 17
6.2.4 Normalized constraints . . . 19
6.2.5 Linking . . . 19

7 Other tools **19**
7.1 Benchmarking . . . 19
7.2 Delta debugging . . . 19
7.2.1 SMT-RAT's own delta tool . . . 19
7.2.2 Using ddSMT . . . 20

7.3 Analyzer . . . 20
7.4 Preprocessing . . . 20
7.5 Benchmax . . . 20
7.5.1 General usage . . . 20
7.5.2 Tools . . . 21
7.5.3 Backends . . . 21
7.5.4 Troubleshooting . . . 22
7.5.5 Benchmax python utility . . . 22
7.6 Delta . . . 23
7.6.1 Delta debugging . . . 23

8 Developers information **23**
8.1 Code style . . . 24
8.2 Documentation . . . 24
8.2.1 Code comments . . . 24
8.2.2 Writing out-of-source documentation . . . 25
8.3 Settings . . . 25
8.4 Logging . . . 26
8.5 Statistics and timing . . . 26
8.6 Testing . . . 27
8.7 Validation . . . 27
8.8 Checkpoints . . . 28
8.9 Finding and Reporting Bugs . . . 29

9 README **29**

10 NewCoveringModule #and **29**
10.1 Introduction . . . 29
10.2 Usage . . . 30
10.3 Efficiency . . . 30

11 Todo List **30**

12 Hierarchical Index **30**
12.1 Class Hierarchy . . . 30

13 Data Structure Index **56**
13.1 Data Structures . . . 56

14 Namespace Documentation **81**
14.1 benchmax Namespace Reference . . . 81
14.1.1 Typedef Documentation . . . 84
14.1.2 Function Documentation . . . 84
14.2 benchmax::settings Namespace Reference . . . 88
14.2.1 Function Documentation . . . 88

14.3 benchmax::slurm Namespace Reference . . . 89
14.3.1 Function Documentation . . . 90
14.4 benchmax::ssh Namespace Reference . . . 92
14.5 carl Namespace Reference . . . 92
14.6 delta Namespace Reference . . . 92
14.6.1 Typedef Documentation . . . 94
14.6.2 Function Documentation . . . 94
14.7 Minisat Namespace Reference . . . 96
14.7.1 Typedef Documentation . . . 98
14.7.2 Function Documentation . . . 98
14.7.3 Variable Documentation . . . 102
14.8 smtrat Namespace Reference . . . 102
14.8.1 Detailed Description . . . 111
14.8.2 Typedef Documentation . . . 113
14.8.3 Enumeration Type Documentation . . . 116
14.8.4 Function Documentation . . . 118
14.8.5 Variable Documentation . . . 132
14.9 smtrat::analyzer Namespace Reference . . . 133
14.9.1 Function Documentation . . . 134
14.10 smtrat::cad Namespace Reference . . . 135
14.10.1 Typedef Documentation . . . 137
14.10.2 Enumeration Type Documentation . . . 137
14.10.3 Function Documentation . . . 139
14.11 smtrat::cad::debug Namespace Reference . . . 142
14.12 smtrat::cad::full Namespace Reference . . . 142
14.12.1 Typedef Documentation . . . 142
14.13 smtrat::cad::full_ec Namespace Reference . . . 142
14.13.1 Typedef Documentation . . . 142
14.14 smtrat::cad::preprocessor Namespace Reference . . . 142
14.14.1 Function Documentation . . . 142
14.15 smtrat::cad::projection Namespace Reference . . . 142
14.15.1 Function Documentation . . . 143
14.16 smtrat::cad::projection::brown Namespace Reference . . . 145
14.16.1 Detailed Description . . . 145
14.16.2 Function Documentation . . . 145
14.17 smtrat::cad::projection::collins Namespace Reference . . . 145
14.17.1 Detailed Description . . . 146
14.17.2 Function Documentation . . . 146
14.18 smtrat::cad::projection::hong Namespace Reference . . . 146
14.18.1 Detailed Description . . . 146
14.18.2 Function Documentation . . . 146
14.19 smtrat::cad::projection::lazard Namespace Reference . . . 147

14.19.1 Detailed Description . . . 147
14.19.2 Function Documentation . . . 147
14.20 smtrat::cad::projection::mccallum Namespace Reference . . . 147
14.20.1 Detailed Description . . . 148
14.20.2 Function Documentation . . . 148
14.21 smtrat::cad::projection::mccallum_partial Namespace Reference . . . 148
14.21.1 Detailed Description . . . 148
14.21.2 Function Documentation . . . 148
14.22 smtrat::cad::projection_compare Namespace Reference . . . 149
14.22.1 Typedef Documentation . . . 149
14.22.2 Function Documentation . . . 149
14.23 smtrat::cad::sample_compare Namespace Reference . . . 150
14.23.1 Detailed Description . . . 151
14.23.2 Typedef Documentation . . . 151
14.23.3 Function Documentation . . . 151
14.24 smtrat::cad::variable_ordering Namespace Reference . . . 152
14.24.1 Function Documentation . . . 152
14.25 smtrat::cadcells Namespace Reference . . . 152
14.25.1 Detailed Description . . . 153
14.25.2 Typedef Documentation . . . 153
14.25.3 Variable Documentation . . . 153
14.26 smtrat::cadcells::algorithms Namespace Reference . . . 154
14.26.1 Detailed Description . . . 154
14.26.2 Typedef Documentation . . . 154
14.26.3 Function Documentation . . . 154
14.26.4 Variable Documentation . . . 156
14.27 smtrat::cadcells::algorithms::helper Namespace Reference . . . 157
14.27.1 Function Documentation . . . 157
14.28 smtrat::cadcells::datastructures Namespace Reference . . . 157
14.28.1 Detailed Description . . . 159
14.28.2 Typedef Documentation . . . 161
14.28.3 Enumeration Type Documentation . . . 161
14.28.4 Function Documentation . . . 161
14.29 smtrat::cadcells::datastructures::detail Namespace Reference . . . 165
14.30 smtrat::cadcells::helper Namespace Reference . . . 165
14.30.1 Function Documentation . . . 165
14.31 smtrat::cadcells::operators Namespace Reference . . . 165
14.31.1 Enumeration Type Documentation . . . 166
14.31.2 Function Documentation . . . 166
14.32 smtrat::cadcells::operators::delineation Namespace Reference . . . 168
14.32.1 Function Documentation . . . 168
14.33 smtrat::cadcells::operators::properties Namespace Reference . . . 169

14.33.1 Function Documentation . . . 169
14.34 smtrat::cadcells::operators::rules Namespace Reference . . . 171
14.34.1 Function Documentation . . . 172
14.35 smtrat::cadcells::representation Namespace Reference . . . 174
14.35.1 Enumeration Type Documentation . . . 175
14.35.2 Function Documentation . . . 175
14.36 smtrat::cadcells::representation::util Namespace Reference . . . 177
14.36.1 Function Documentation . . . 177
14.37 smtrat::compile_information Namespace Reference . . . 177
14.37.1 Detailed Description . . . 178
14.37.2 Variable Documentation . . . 178
14.38 smtrat::datastructures Namespace Reference . . . 179
14.39 smtrat::execution Namespace Reference . . . 179
14.39.1 Enumeration Type Documentation . . . 179
14.40 smtrat::expression Namespace Reference . . . 179
14.40.1 Typedef Documentation . . . 180
14.40.2 Enumeration Type Documentation . . . 180
14.40.3 Function Documentation . . . 181
14.41 smtrat::expression::simplifier Namespace Reference . . . 182
14.41.1 Typedef Documentation . . . 182
14.42 smtrat::groebner Namespace Reference . . . 182
14.42.1 Typedef Documentation . . . 183
14.42.2 Function Documentation . . . 183
14.43 smtrat::icp Namespace Reference . . . 183
14.43.1 Typedef Documentation . . . 183
14.43.2 Enumeration Type Documentation . . . 183
14.43.3 Function Documentation . . . 184
14.44 smtrat::impl Namespace Reference . . . 185
14.44.1 Function Documentation . . . 185
14.44.2 Variable Documentation . . . 185
14.45 smtrat::lra Namespace Reference . . . 185
14.45.1 Typedef Documentation . . . 186
14.45.2 Function Documentation . . . 186
14.45.3 Variable Documentation . . . 191
14.46 smtrat::lve Namespace Reference . . . 191
14.46.1 Function Documentation . . . 192
14.47 smtrat::maxsmt Namespace Reference . . . 192
14.47.1 Detailed Description . . . 192
14.48 smtrat::mcsat Namespace Reference . . . 192
14.48.1 Typedef Documentation . . . 193
14.48.2 Enumeration Type Documentation . . . 194
14.48.3 Function Documentation . . . 194

14.49 smtrat::mcsat::arithmetic Namespace Reference . . . 196
14.49.1 Function Documentation . . . 196
14.50 smtrat::mcsat::constraint_type Namespace Reference . . . 196
14.50.1 Function Documentation . . . 196
14.51 smtrat::mcsat::fm Namespace Reference . . . 197
14.51.1 Detailed Description . . . 198
14.51.2 Function Documentation . . . 198
14.52 smtrat::mcsat::icp Namespace Reference . . . 198
14.53 smtrat::mcsat::nlsat Namespace Reference . . . 198
14.54 smtrat::mcsat::nlsat::helper Namespace Reference . . . 199
14.54.1 Function Documentation . . . 199
14.55 smtrat::mcsat::onecell Namespace Reference . . . 200
14.56 smtrat::mcsat::onecellcad Namespace Reference . . . 200
14.56.1 Typedef Documentation . . . 201
14.56.2 Enumeration Type Documentation . . . 201
14.56.3 Function Documentation . . . 202
14.57 smtrat::mcsat::onecellcad::firstlevel Namespace Reference . . . 206
14.57.1 Typedef Documentation . . . 207
14.57.2 Function Documentation . . . 207
14.58 smtrat::mcsat::onecellcad::levelwise Namespace Reference . . . 210
14.59 smtrat::mcsat::onecellcad::recursive Namespace Reference . . . 210
14.59.1 Detailed Description . . . 210
14.59.2 Enumeration Type Documentation . . . 211
14.59.3 Function Documentation . . . 211
14.59.4 Variable Documentation . . . 211
14.60 smtrat::mcsat::smtaf Namespace Reference . . . 211
14.60.1 Typedef Documentation . . . 212
14.60.2 Function Documentation . . . 212
14.61 smtrat::mcsat::variableordering Namespace Reference . . . 212
14.61.1 Function Documentation . . . 212
14.62 smtrat::mcsat::variableordering::detail Namespace Reference . . . 213
14.62.1 Function Documentation . . . 213
14.63 smtrat::mcsat::vs Namespace Reference . . . 215
14.64 smtrat::mcsat::vs::helper Namespace Reference . . . 215
14.64.1 Function Documentation . . . 215
14.65 smtrat::onecellcad Namespace Reference . . . 216
14.66 smtrat::onecellcad::recursive Namespace Reference . . . 216
14.66.1 Typedef Documentation . . . 217
14.66.2 Function Documentation . . . 217
14.67 smtrat::options_detail Namespace Reference . . . 219
14.67.1 Function Documentation . . . 219
14.68 smtrat::parseformula Namespace Reference . . . 219

14.69 smtrat::parser Namespace Reference . . . 220
14.69.1 Typedef Documentation . . . 221
14.69.2 Enumeration Type Documentation . . . 221
14.69.3 Function Documentation . . . 222
14.70 smtrat::parser::arithmetic Namespace Reference . . . 223
14.70.1 Typedef Documentation . . . 223
14.70.2 Function Documentation . . . 223
14.71 smtrat::parser::conversion Namespace Reference . . . 223
14.72 smtrat::parser::core Namespace Reference . . . 223
14.72.1 Function Documentation . . . 223
14.73 smtrat::parser::types Namespace Reference . . . 224
14.73.1 Typedef Documentation . . . 225
14.74 smtrat::parser::uninterpreted Namespace Reference . . . 226
14.74.1 Function Documentation . . . 226
14.75 smtrat::preprocessor Namespace Reference . . . 226
14.76 smtrat::qe Namespace Reference . . . 226
14.76.1 Typedef Documentation . . . 226
14.76.2 Enumeration Type Documentation . . . 226
14.76.3 Function Documentation . . . 227
14.77 smtrat::qe::cad Namespace Reference . . . 227
14.77.1 Function Documentation . . . 227
14.78 smtrat::qe::fm Namespace Reference . . . 228
14.78.1 Function Documentation . . . 228
14.79 smtrat::resource Namespace Reference . . . 228
14.79.1 Function Documentation . . . 228
14.80 smtrat::sat Namespace Reference . . . 229
14.81 smtrat::sat::detail Namespace Reference . . . 229
14.81.1 Function Documentation . . . 229
14.82 smtrat::settings Namespace Reference . . . 229
14.83 smtrat::statistics Namespace Reference . . . 229
14.83.1 Function Documentation . . . 229
14.84 smtrat::uf_impl Namespace Reference . . . 230
14.85 smtrat::unsatcore Namespace Reference . . . 230
14.85.1 Detailed Description . . . 230
14.86 smtrat::validation Namespace Reference . . . 230
14.86.1 Enumeration Type Documentation . . . 230
14.86.2 Function Documentation . . . 231
14.87 smtrat::vb Namespace Reference . . . 231
14.87.1 Function Documentation . . . 231
14.88 smtrat::vs Namespace Reference . . . 232
14.88.1 Typedef Documentation . . . 234
14.88.2 Function Documentation . . . 235

15 Data Structure Documentation 245
15.1 smtrat::AbstractModuleFactory Struct Reference . . . 245
15.1.1 Constructor & Destructor Documentation . . . 246
15.1.2 Member Function Documentation . . . 246
15.2 smtrat::parser::AbstractTheory Struct Reference . . . 246
15.2.1 Detailed Description . . . 247
15.2.2 Constructor & Destructor Documentation . . . 247
15.2.3 Member Function Documentation . . . 247
15.2.4 Field Documentation . . . 249
15.3 smtrat::cad::sample_compare::absvalue Struct Reference . . . 249
15.4 smtrat::analyzer::AnalysisSettings Struct Reference . . . 249
15.4.1 Field Documentation . . . 249
15.5 smtrat::analyzer::AnalyzerStatistics Struct Reference . . . 249
15.5.1 Member Function Documentation . . . 249
15.6 smtrat::AnnotatedBVTerm Class Reference . . . 250
15.6.1 Constructor & Destructor Documentation . . . 250
15.6.2 Member Function Documentation . . . 250
15.6.3 Friends And Related Function Documentation . . . 250
15.7 benchmax::slurm::ArchiveProperties Struct Reference . . . 251
15.7.1 Detailed Description . . . 251
15.7.2 Field Documentation . . . 251
15.8 smtrat::parser::ArithmeticTheory Struct Reference . . . 251
15.8.1 Detailed Description . . . 252
15.8.2 Constructor & Destructor Documentation . . . 252
15.8.3 Member Function Documentation . . . 252
15.8.4 Field Documentation . . . 253
15.9 smtrat::parser::types::ArithmeticTheory Struct Reference . . . 254
15.9.1 Detailed Description . . . 254
15.9.2 Member Typedef Documentation . . . 254
15.10 smtrat::execution::Assertion Struct Reference . . . 254
15.10.1 Field Documentation . . . 254
15.11 smtrat::cad::preprocessor::AssignmentCollector Class Reference . . . 255
15.11.1 Member Typedef Documentation . . . 255
15.11.2 Constructor & Destructor Documentation . . . 255
15.11.3 Member Function Documentation . . . 255
15.12 smtrat::mcsat::arithmetic::AssignmentFinder Struct Reference . . . 256
15.12.1 Member Function Documentation . . . 256
15.13 smtrat::mcsat::smtaf::AssignmentFinder< Settings > Struct Template Reference . . . 256
15.13.1 Member Function Documentation . . . 256
15.14 smtrat::mcsat::arithmetic::AssignmentFinder_detail Class Reference . . . 256
15.14.1 Member Typedef Documentation . . . 257
15.14.2 Constructor & Destructor Documentation . . . 257

15.14.3 Member Function Documentation . . . 257
15.15 smtrat::mcsat::smtaf::AssignmentFinder_SMT Class Reference . . . 257
15.15.1 Constructor & Destructor Documentation . . . 257
15.15.2 Member Function Documentation . . . 257
15.16 smtrat::cadcells::datastructures::detail::AssignmentProperties Struct Reference . . . 258
15.16.1 Field Documentation . . . 258
15.17 smtrat::parser::Attribute Class Reference . . . 258
15.17.1 Detailed Description . . . 258
15.17.2 Member Typedef Documentation . . . 258
15.17.3 Constructor & Destructor Documentation . . . 259
15.17.4 Member Function Documentation . . . 259
15.17.5 Field Documentation . . . 259
15.18 smtrat::parser::AttributeParser Struct Reference . . . 259
15.18.1 Constructor & Destructor Documentation . . . 259
15.18.2 Field Documentation . . . 259
15.19 smtrat::parser::AttributeValueParser Struct Reference . . . 260
15.19.1 Member Typedef Documentation . . . 260
15.19.2 Constructor & Destructor Documentation . . . 260
15.19.3 Field Documentation . . . 260
15.20 smtrat::AxiomFactory Class Reference . . . 260
15.20.1 Member Enumeration Documentation . . . 260
15.20.2 Member Function Documentation . . . 261
15.21 benchmax::Backend Class Reference . . . 261
15.21.1 Detailed Description . . . 262
15.21.2 Constructor & Destructor Documentation . . . 262
15.21.3 Member Function Documentation . . . 262
15.21.4 Field Documentation . . . 263
15.22 smtrat::Backend< Settings > Class Template Reference . . . 263
15.22.1 Constructor & Destructor Documentation . . . 264
15.22.2 Member Function Documentation . . . 264
15.23 smtrat::BackendLink Class Reference . . . 266
15.23.1 Constructor & Destructor Documentation . . . 266
15.23.2 Member Function Documentation . . . 266
15.24 smtrat::BackendSynchronisation Class Reference . . . 267
15.24.1 Constructor & Destructor Documentation . . . 267
15.24.2 Member Function Documentation . . . 267
15.25 smtrat::Backtrackable< UnionFind > Struct Template Reference . . . 267
15.25.1 Member Typedef Documentation . . . 268
15.25.2 Constructor & Destructor Documentation . . . 268
15.25.3 Member Function Documentation . . . 268
15.25.4 Field Documentation . . . 269
15.26 smtrat::cadcells::datastructures::BaseDerivation< Properties > Class Template Reference . . . 269

15.26.1 Detailed Description . . . 269
15.26.2 Constructor & Destructor Documentation . . . 270
15.26.3 Member Function Documentation . . . 270
15.26.4 Friends And Related Function Documentation . . . 271
15.27 smtrat::cad::BaseProjection< Settings > Class Template Reference . . . 271
15.27.1 Member Typedef Documentation . . . 272
15.27.2 Constructor & Destructor Documentation . . . 272
15.27.3 Member Function Documentation . . . 272
15.27.4 Field Documentation . . . 277
15.28 smtrat::cad::BaseSettings Struct Reference . . . 277
15.28.1 Field Documentation . . . 277
15.29 smtrat::expression::simplifier::BaseSimplifier Struct Reference . . . 278
15.29.1 Member Function Documentation . . . 279
15.30 smtrat::cad::Origin::BaseType Struct Reference . . . 280
15.30.1 Constructor & Destructor Documentation . . . 280
15.30.2 Member Function Documentation . . . 281
15.30.3 Friends And Related Function Documentation . . . 281
15.30.4 Field Documentation . . . 281
15.31 smtrat::BEModule< Settings > Class Template Reference . . . 282
15.31.1 Member Typedef Documentation . . . 287
15.31.2 Member Enumeration Documentation . . . 287
15.31.3 Constructor & Destructor Documentation . . . 287
15.31.4 Member Function Documentation . . . 287
15.31.5 Field Documentation . . . 308
15.32 benchmax::BenchmarkResult Struct Reference . . . 311
15.32.1 Detailed Description . . . 312
15.32.2 Member Function Documentation . . . 312
15.32.3 Field Documentation . . . 312
15.33 benchmax::BenchmarkSet Class Reference . . . 313
15.33.1 Detailed Description . . . 313
15.33.2 Member Function Documentation . . . 314
15.34 benchmax::settings::BenchmarkSettings Struct Reference . . . 314
15.34.1 Detailed Description . . . 314
15.34.2 Field Documentation . . . 314
15.35 smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName > Class Template Reference . . . 315
15.35.1 Detailed Description . . . 316
15.35.2 Member Typedef Documentation . . . 316
15.35.3 Member Function Documentation . . . 316
15.36 smtrat::expression::BinaryExpression Struct Reference . . . 317
15.36.1 Constructor & Destructor Documentation . . . 317
15.36.2 Field Documentation . . . 318

15.37 smtrat::datastructures::BinaryHeap< T > Class Template Reference . . . 318
15.37.1 Constructor & Destructor Documentation . . . 318
15.37.2 Member Function Documentation . . . 318
15.38 smtrat::parser::BinaryParser Struct Reference . . . 319
15.38.1 Detailed Description . . . 319
15.38.2 Member Typedef Documentation . . . 319
15.38.3 Constructor & Destructor Documentation . . . 319
15.38.4 Member Function Documentation . . . 319
15.38.5 Field Documentation . . . 319
15.39 smtrat::parser::BitvectorTheory Struct Reference . . . 320
15.39.1 Detailed Description . . . 320
15.39.2 Member Typedef Documentation . . . 320
15.39.3 Constructor & Destructor Documentation . . . 320
15.39.4 Member Function Documentation . . . 320
15.39.5 Field Documentation . . . 322
15.40 smtrat::parser::types::BitvectorTheory Struct Reference . . . 323
15.40.1 Detailed Description . . . 323
15.40.2 Member Typedef Documentation . . . 323
15.41 smtrat::BlastedConstr Class Reference . . . 323
15.41.1 Constructor & Destructor Documentation . . . 323
15.41.2 Member Function Documentation . . . 324
15.41.3 Friends And Related Function Documentation . . . 324
15.42 smtrat::BlastedPoly Class Reference . . . 324
15.42.1 Constructor & Destructor Documentation . . . 324
15.42.2 Member Function Documentation . . . 325
15.42.3 Friends And Related Function Documentation . . . 325
15.43 smtrat::mcsat::Bookkeeping Class Reference . . . 325
15.43.1 Detailed Description . . . 326
15.43.2 Member Function Documentation . . . 326
15.44 smtrat::parser::BooleanEncodingTheory Struct Reference . . . 327
15.44.1 Detailed Description . . . 327
15.44.2 Constructor & Destructor Documentation . . . 327
15.44.3 Member Function Documentation . . . 328
15.44.4 Field Documentation . . . 329
15.45 Minisat::BoolOption Class Reference . . . 329
15.45.1 Constructor & Destructor Documentation . . . 330
15.45.2 Member Function Documentation . . . 330
15.45.3 Field Documentation . . . 330
15.46 smtrat::BoolUEQRewriter Struct Reference . . . 331
15.46.1 Constructor & Destructor Documentation . . . 331
15.46.2 Member Function Documentation . . . 331
15.47 smtrat::lra::Bound< T1, T2 > Class Template Reference . . . 331

15.47.1 Detailed Description . . . 332
15.47.2 Member Typedef Documentation . . . 333
15.47.3 Member Enumeration Documentation . . . 333
15.47.4 Constructor & Destructor Documentation . . . 333
15.47.5 Member Function Documentation . . . 334
15.47.6 Friends And Related Function Documentation . . . 341
15.48 smtrat::vb::Bound< T > Class Template Reference . . . 342
15.48.1 Detailed Description . . . 342
15.48.2 Member Enumeration Documentation . . . 342
15.48.3 Constructor & Destructor Documentation . . . 343
15.48.4 Member Function Documentation . . . 343
15.48.5 Friends And Related Function Documentation . . . 346
15.49 smtrat::mcsat::fm::Bound Struct Reference . . . 346
15.49.1 Constructor & Destructor Documentation . . . 346
15.49.2 Friends And Related Function Documentation . . . 346
15.49.3 Field Documentation . . . 347
15.50 smtrat::Branching Struct Reference . . . 347
15.50.1 Detailed Description . . . 347
15.50.2 Constructor & Destructor Documentation . . . 347
15.50.3 Field Documentation . . . 348
15.51 smtrat::BVAnnotation Class Reference . . . 348
15.51.1 Constructor & Destructor Documentation . . . 348
15.51.2 Member Function Documentation . . . 349
15.51.3 Friends And Related Function Documentation . . . 350
15.52 smtrat::BVDirectEncoder Class Reference . . . 350
15.52.1 Constructor & Destructor Documentation . . . 350
15.52.2 Member Function Documentation . . . 350
15.53 smtrat::BVModule< Settings > Class Template Reference . . . 350
15.53.1 Member Typedef Documentation . . . 356
15.53.2 Member Enumeration Documentation . . . 356
15.53.3 Constructor & Destructor Documentation . . . 356
15.53.4 Member Function Documentation . . . 356
15.53.5 Field Documentation . . . 377
15.54 smtrat::by_address_hasher< IterType > Struct Template Reference . . . 379
15.54.1 Detailed Description . . . 380
15.54.2 Member Typedef Documentation . . . 380
15.54.3 Member Function Documentation . . . 380
15.55 smtrat::cad::CAD< Settings > Class Template Reference . . . 380
15.55.1 Member Typedef Documentation . . . 381
15.55.2 Constructor & Destructor Documentation . . . 381
15.55.3 Member Function Documentation . . . 381
15.55.4 Friends And Related Function Documentation . . . 382

15.55.5 Field Documentation . . . 383
15.56 smtrat::qe::cad::CAD< Settings > Class Template Reference . . . 383
15.56.1 Member Typedef Documentation . . . 383
15.56.2 Constructor & Destructor Documentation . . . 383
15.56.3 Member Function Documentation . . . 383
15.57 smtrat::cad::CADConstraints< BT > Class Template Reference . . . 384
15.57.1 Member Typedef Documentation . . . 385
15.57.2 Constructor & Destructor Documentation . . . 385
15.57.3 Member Function Documentation . . . 386
15.57.4 Friends And Related Function Documentation . . . 387
15.57.5 Field Documentation . . . 387
15.58 smtrat::cad::CADCore< CH > Struct Template Reference . . . 388
15.59 smtrat::cad::CADCore< CoreHeuristic::BySample > Struct Template Reference . . . 388
15.59.1 Member Function Documentation . . . 389
15.60 smtrat::cad::CADCore< CoreHeuristic::EnumerateAll > Struct Template Reference . . . 389
15.60.1 Member Function Documentation . . . 389
15.61 smtrat::cad::CADCore< CoreHeuristic::Interleave > Struct Template Reference . . . 389
15.61.1 Member Function Documentation . . . 389
15.62 smtrat::cad::CADCore< CoreHeuristic::PreferProjection > Struct Template Reference . . . 390
15.62.1 Member Function Documentation . . . 390
15.63 smtrat::cad::CADCore< CoreHeuristic::PreferSampling > Struct Template Reference . . . 390
15.63.1 Member Function Documentation . . . 390
15.64 smtrat::qe::cad::CADElimination Class Reference . . . 390
15.64.1 Constructor & Destructor Documentation . . . 390
15.64.2 Member Function Documentation . . . 390
15.65 smtrat::cad::CADPreprocessor Class Reference . . . 391
15.65.1 Constructor & Destructor Documentation . . . 391
15.65.2 Member Function Documentation . . . 391
15.65.3 Friends And Related Function Documentation . . . 392
15.66 smtrat::cad::CADPreprocessorSettings Struct Reference . . . 392
15.66.1 Member Function Documentation . . . 392
15.66.2 Field Documentation . . . 392
15.67 smtrat::qe::cad::CADSettings Struct Reference . . . 393
15.67.1 Field Documentation . . . 393
15.68 smtrat::CardinalityEncoder Class Reference . . . 394
15.68.1 Constructor & Destructor Documentation . . . 394
15.68.2 Member Function Documentation . . . 394
15.68.3 Field Documentation . . . 395
15.69 smtrat::cadcells::representation::cell< H > Struct Template Reference . . . 395
15.69.1 Detailed Description . . . 395
15.69.2 Member Function Documentation . . . 395
15.70 smtrat::cadcells::representation::cell< CellHeuristic::BIGGEST_CELL > Struct Template Reference 395

15.70.1 Member Function Documentation . . . 396
15.71 smtrat::cadcells::representation::cell< CellHeuristic::CHAIN_EQ > Struct Template Reference . . 396
15.71.1 Member Function Documentation . . . 396
15.72 smtrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS > Struct Template Reference . . . 396
15.72.1 Member Function Documentation . . . 396
15.73 smtrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS_EQ > Struct Template Reference . . . 396
15.73.1 Member Function Documentation . . . 396
15.74 smtrat::cadcells::operators::properties::cell_connected Struct Reference . . . 397
15.74.1 Member Function Documentation . . . 397
15.74.2 Field Documentation . . . 397
15.75 smtrat::cadcells::datastructures::CellDescription Class Reference . . . 397
15.75.1 Detailed Description . . . 398
15.75.2 Constructor & Destructor Documentation . . . 398
15.75.3 Member Function Documentation . . . 398
15.76 smtrat::cadcells::datastructures::CellRepresentation< P > Struct Template Reference . . . 399
15.76.1 Detailed Description . . . 399
15.76.2 Constructor & Destructor Documentation . . . 399
15.76.3 Field Documentation . . . 400
15.77 delta::Checker Class Reference . . . 400
15.77.1 Detailed Description . . . 400
15.77.2 Constructor & Destructor Documentation . . . 400
15.77.3 Member Function Documentation . . . 401
15.78 benchmax::slurm::ChunkedSubmitfileProperties Struct Reference . . . 401
15.78.1 Detailed Description . . . 402
15.78.2 Field Documentation . . . 402
15.79 Minisat::Clause Class Reference . . . 403
15.79.1 Member Function Documentation . . . 403
15.79.2 Friends And Related Function Documentation . . . 405
15.79.3 Field Documentation . . . 405
15.80 Minisat::ClauseAllocator Class Reference . . . 406
15.80.1 Member Typedef Documentation . . . 406
15.80.2 Member Enumeration Documentation . . . 406
15.80.3 Constructor & Destructor Documentation . . . 407
15.80.4 Member Function Documentation . . . 407
15.80.5 Field Documentation . . . 408
15.81 smtrat::mcsat::ClauseChain Class Reference . . . 408
15.81.1 Detailed Description . . . 409
15.81.2 Member Typedef Documentation . . . 409
15.81.3 Constructor & Destructor Documentation . . . 409
15.81.4 Member Function Documentation . . . 409
15.81.5 Friends And Related Function Documentation . . . 411

15.82 smtrat::sat::detail::ClauseChecker Struct Reference . . . 411
15.82.1 Member Function Documentation . . . 411
15.83 smtrat::CMakeOptionPrinter Struct Reference . . . 411
15.83.1 Field Documentation . . . 411
15.84 Minisat::CMap< T > Class Template Reference . . . 411
15.84.1 Member Function Documentation . . . 412
15.85 smtrat::CNFerModule Class Reference . . . 413
15.85.1 Member Enumeration Documentation . . . 418
15.85.2 Constructor & Destructor Documentation . . . 418
15.85.3 Member Function Documentation . . . 419
15.85.4 Field Documentation . . . 441
15.86 smtrat::CoCoAGBModule< Settings > Class Template Reference . . . 444
15.86.1 Member Enumeration Documentation . . . 449
15.86.2 Member Function Documentation . . . 450
15.86.3 Field Documentation . . . 471
15.87 smtrat::CollectBoolsInUEQs Struct Reference . . . 474
15.87.1 Constructor & Destructor Documentation . . . 475
15.87.2 Member Function Documentation . . . 475
15.87.3 Friends And Related Function Documentation . . . 475
15.88 smtrat::CollectionWithOrigins< Element, Origin > Class Template Reference . . . 475
15.88.1 Member Typedef Documentation . . . 475
15.88.2 Constructor & Destructor Documentation . . . 476
15.88.3 Member Function Documentation . . . 476
15.89 smtrat::ICPModule< Settings >::comp Struct Reference . . . 477
15.89.1 Detailed Description . . . 477
15.89.2 Member Function Documentation . . . 477
15.90 smtrat::vs::Condition Class Reference . . . 477
15.90.1 Constructor & Destructor Documentation . . . 478
15.90.2 Member Function Documentation . . . 478
15.90.3 Friends And Related Function Documentation . . . 480
15.91 benchmax::CondorBackend Class Reference . . . 480
15.91.1 Detailed Description . . . 481
15.91.2 Member Function Documentation . . . 481
15.91.3 Field Documentation . . . 482
15.92 smtrat::mcsat::fm::ConflictGenerator< Comparator > Struct Template Reference . . . 482
15.92.1 Constructor & Destructor Documentation . . . 482
15.92.2 Member Function Documentation . . . 482
15.93 smtrat::cad::ConflictGraph Class Reference . . . 483
15.93.1 Detailed Description . . . 483
15.93.2 Constructor & Destructor Documentation . . . 483
15.93.3 Member Function Documentation . . . 483
15.93.4 Friends And Related Function Documentation . . . 484

15.94 smtrat::parser::Theories::ConstantAdder Struct Reference . . . 485
15.94.1 Detailed Description . . . 485
15.94.2 Constructor & Destructor Documentation . . . 485
15.94.3 Member Function Documentation . . . 485
15.94.4 Field Documentation . . . 485
15.95 smtrat::cad::CADConstraints< BT >::ConstraintComparator Struct Reference . . . 485
15.95.1 Member Function Documentation . . . 485
15.96 smtrat::cad::preprocessor::ConstraintUpdate Struct Reference . . . 485
15.96.1 Field Documentation . . . 486
15.97 smtrat::ConstrTree Class Reference . . . 486
15.97.1 Constructor & Destructor Documentation . . . 486
15.97.2 Member Function Documentation . . . 486
15.98 delta::Consumer Class Reference . . . 486
15.98.1 Detailed Description . . . 487
15.98.2 Constructor & Destructor Documentation . . . 487
15.98.3 Member Function Documentation . . . 487
15.99 smtrat::LRAModule< Settings >::Context Struct Reference . . . 488
15.99.1 Detailed Description . . . 489
15.99.2 Constructor & Destructor Documentation . . . 489
15.99.3 Field Documentation . . . 489
15.100 smtrat::icp::ContractionCandidate Class Reference . . . 489
15.100.1 Constructor & Destructor Documentation . . . 490
15.100.2 Member Function Documentation . . . 491
15.101 smtrat::icp::contractionCandidateComp Struct Reference . . . 494
15.101.1 Member Function Documentation . . . 494
15.102 smtrat::icp::ContractionCandidateManager Class Reference . . . 494
15.102.1 Constructor & Destructor Documentation . . . 495
15.102.2 Member Function Documentation . . . 495
15.103 smtrat::parser::conversion::Converter< To > Struct Template Reference . . . 496
15.103.1 Member Function Documentation . . . 497
15.104 smtrat::parser::conversion::Converter< FormulaT > Struct Template Reference . . . 497
15.104.1 Member Function Documentation . . . 497
15.105 smtrat::parser::conversion::Converter< Poly > Struct Template Reference . . . 497
15.105.1 Member Function Documentation . . . 498
15.106 smtrat::parser::conversion::Converter< types::BVTerm > Struct Template Reference . . . 498
15.106.1 Member Function Documentation . . . 498
15.107 benchmax::settings::CoreSettings Struct Reference . . . 499
15.107.1 Detailed Description . . . 499
15.107.2 Field Documentation . . . 499
15.108 smtrat::settings::CoreSettings Struct Reference . . . 500
15.108.1 Field Documentation . . . 500
15.109 smtrat::parser::CoreTheory Struct Reference . . . 501

15.109.1 Detailed Description . . . 501
15.109.2 Member Typedef Documentation . . . 501
15.109.3 Constructor & Destructor Documentation . . . 501
15.109.4 Member Function Documentation . . . 501
15.109.5 Field Documentation . . . 503
15.110 smtrat::parser::types::CoreTheory Struct Reference . . . 503
15.110.1 Detailed Description . . . 503
15.110.2 Member Typedef Documentation . . . 503
15.111 smtrat::cadcells::representation::covering< H > Struct Template Reference . . . 504
15.111.1 Member Function Documentation . . . 504
15.112 smtrat::mcsat::arithmetic::Covering Class Reference . . . 504
15.112.1 Detailed Description . . . 504
15.112.2 Constructor & Destructor Documentation . . . 504
15.112.3 Member Function Documentation . . . 504
15.112.4 Friends And Related Function Documentation . . . 505
15.113 smtrat::cadcells::representation::covering< CoveringHeuristic::CHAIN_COVERING > Struct Template Reference . . . 505
15.113.1 Member Function Documentation . . . 505
15.114 smtrat::cadcells::representation::covering< CoveringHeuristic::DEFAULT_COVERING > Struct Template Reference . . . 505
15.114.1 Member Function Documentation . . . 506
15.115 smtrat::mcsat::onecellcad::firstlevel::covering_at_level Struct Reference . . . 506
15.115.1 Field Documentation . . . 506
15.116 smtrat::cadcells::datastructures::CoveringDescription Class Reference . . . 506
15.116.1 Detailed Description . . . 506
15.116.2 Member Function Documentation . . . 506
15.117 smtrat::cadcells::datastructures::CoveringRepresentation< P > Struct Template Reference . . . 507
15.117.1 Detailed Description . . . 507
15.117.2 Member Function Documentation . . . 507
15.117.3 Field Documentation . . . 507
15.118 smtrat::mcsat::onecellcad::recursive::CoverNullification Struct Reference . . . 508
15.118.1 Field Documentation . . . 508
15.119 smtrat::CSplitModule< Settings > Class Template Reference . . . 508
15.119.1 Member Typedef Documentation . . . 513
15.119.2 Member Enumeration Documentation . . . 513
15.119.3 Constructor & Destructor Documentation . . . 513
15.119.4 Member Function Documentation . . . 513
15.119.5 Field Documentation . . . 535
15.120 smtrat::CubeLIAModule< Settings > Class Template Reference . . . 538
15.120.1 Member Typedef Documentation . . . 543
15.120.2 Member Enumeration Documentation . . . 543
15.120.3 Constructor & Destructor Documentation . . . 544
15.120.4 Member Function Documentation . . . 544

15.120.5 Field Documentation . . . 565
15.121 smtrat::CurryModule< Settings > Class Template Reference . . . 568
15.121.1 Member Typedef Documentation . . . 573
15.121.2 Member Enumeration Documentation . . . 573
15.121.3 Constructor & Destructor Documentation . . . 574
15.121.4 Member Function Documentation . . . 574
15.121.5 Field Documentation . . . 595
15.122 smtrat::CycleEnumerator< FHG, Collector > Struct Template Reference . . . 598
15.122.1 Detailed Description . . . 598
15.122.2 Constructor & Destructor Documentation . . . 598
15.122.3 Member Function Documentation . . . 598
15.123 benchmax::Database Class Reference . . . 598
15.123.1 Detailed Description . . . 599
15.123.2 Member Typedef Documentation . . . 599
15.123.3 Member Function Documentation . . . 599
15.124 benchmax::DBAL Class Reference . . . 600
15.124.1 Member Typedef Documentation . . . 601
15.124.2 Member Function Documentation . . . 601
15.125 smtrat::parser::DecimalParser Struct Reference . . . 604
15.125.1 Detailed Description . . . 604
15.126 Minisat::DeepEqual< K > Struct Template Reference . . . 604
15.126.1 Member Function Documentation . . . 604
15.127 Minisat::DeepHash< K > Struct Template Reference . . . 604
15.127.1 Member Function Documentation . . . 604
15.128 smtrat::mcsat::fm::DefaultComparator Struct Reference . . . 605
15.128.1 Detailed Description . . . 605
15.128.2 Member Function Documentation . . . 605
15.128.3 Field Documentation . . . 605
15.129 smtrat::mcsat::vs::DefaultSettings Struct Reference . . . 605
15.129.1 Field Documentation . . . 605
15.130 smtrat::mcsat::smtaf::DefaultSettings Struct Reference . . . 605
15.130.1 Field Documentation . . . 605
15.131 smtrat::mcsat::fm::DefaultSettings Struct Reference . . . 606
15.131.1 Field Documentation . . . 606
15.132 smtrat::cad::projection_compare::degree Struct Reference . . . 606
15.133 smtrat::mcsat::onecellcad::recursive::DegreeAscending Struct Reference . . . 606
15.133.1 Field Documentation . . . 606
15.134 smtrat::analyzer::DegreeCollector Struct Reference . . . 606
15.134.1 Member Function Documentation . . . 606
15.134.2 Field Documentation . . . 607
15.135 smtrat::mcsat::onecellcad::recursive::DegreeDescending Struct Reference . . . 607
15.135.1 Field Documentation . . . 607

15.136 smtrat::cadcells::datastructures::DelineatedDerivation< Properties > Class Template Reference 607
15.136.1 Detailed Description . . . 607
15.136.2 Constructor & Destructor Documentation . . . 608
15.136.3 Member Function Documentation . . . 608
15.137 smtrat::cadcells::datastructures::Delineation Class Reference . . . 609
15.137.1 Detailed Description . . . 610
15.137.2 Constructor & Destructor Documentation . . . 610
15.137.3 Member Function Documentation . . . 610
15.137.4 Friends And Related Function Documentation . . . 611
15.138 smtrat::cadcells::representation::delineation< H > Struct Template Reference . . . 611
15.138.1 Member Function Documentation . . . 611
15.139 smtrat::cadcells::representation::delineation< DelineationHeuristic::CHAIN > Struct Template Reference . . . 611
15.139.1 Member Function Documentation . . . 611
15.140 smtrat::cadcells::datastructures::DelineationInterval Class Reference . . . 611
15.140.1 Detailed Description . . . 612
15.140.2 Member Function Documentation . . . 612
15.140.3 Friends And Related Function Documentation . . . 612
15.141 smtrat::cadcells::datastructures::DelineationRepresentation< P > Struct Template Reference . . . 612
15.141.1 Detailed Description . . . 612
15.141.2 Constructor & Destructor Documentation . . . 613
15.141.3 Field Documentation . . . 613
15.142 smtrat::mcsat::icp::Dependencies Class Reference . . . 613
15.142.1 Member Function Documentation . . . 613
15.143 smtrat::cadcells::datastructures::DerivationRef< Properties > Class Template Reference . . . 614
15.143.1 Detailed Description . . . 614
15.143.2 Constructor & Destructor Documentation . . . 614
15.143.3 Member Function Documentation . . . 615
15.143.4 Friends And Related Function Documentation . . . 616
15.144 smtrat::mcsat::onecellcad::recursive::DontCoverNullification Struct Reference . . . 616
15.144.1 Field Documentation . . . 616
15.145 smtrat::cad::debug::DotSubgraph Struct Reference . . . 617
15.145.1 Constructor & Destructor Documentation . . . 617
15.145.2 Member Function Documentation . . . 617
15.145.3 Friends And Related Function Documentation . . . 617
15.145.4 Field Documentation . . . 617
15.146 Minisat::DoubleOption Class Reference . . . 617
15.146.1 Constructor & Destructor Documentation . . . 618
15.146.2 Member Function Documentation . . . 618
15.146.3 Field Documentation . . . 618
15.147 Minisat::DoubleRange Struct Reference . . . 619
15.147.1 Constructor & Destructor Documentation . . . 619

15.147.2 Field Documentation . . . 619
15.148 smtrat::expression::simplifier::DuplicateSimplifier Struct Reference . . . 619
15.148.1 Member Function Documentation . . . 620
15.149 smtrat::DynamicPriorityQueue< T, Compare > Class Template Reference . . . 621
15.149.1 Member Typedef Documentation . . . 622
15.149.2 Constructor & Destructor Documentation . . . 622
15.149.3 Member Function Documentation . . . 622
15.149.4 Field Documentation . . . 623
15.150 smtrat::dynarray< T > Class Template Reference . . . 623
15.150.1 Detailed Description . . . 624
15.150.2 Member Typedef Documentation . . . 624
15.150.3 Constructor & Destructor Documentation . . . 624
15.150.4 Member Function Documentation . . . 625
15.150.5 Field Documentation . . . 627
15.151 smtrat::dynarray_allocator< T > Class Template Reference . . . 627
15.151.1 Friends And Related Function Documentation . . . 627
15.152 smtrat::datastructures::DynHeap< KeyType, Compare, WeightType > Class Template Reference 627
15.152.1 Constructor & Destructor Documentation . . . 627
15.152.2 Member Function Documentation . . . 628
15.153 smtrat::cad::ProjectionGlobalInformation::ECData Struct Reference . . . 628
15.153.1 Field Documentation . . . 628
15.154 smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Edge Struct Reference . . . 629
15.154.1 Detailed Description . . . 629
15.154.2 Constructor & Destructor Documentation . . . 629
15.154.3 Member Function Documentation . . . 629
15.154.4 Field Documentation . . . 630
15.155 smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType > Struct Template Reference 631
15.155.1 Detailed Description . . . 631
15.155.2 Member Typedef Documentation . . . 631
15.155.3 Constructor & Destructor Documentation . . . 631
15.155.4 Member Function Documentation . . . 632
15.155.5 Friends And Related Function Documentation . . . 633
15.155.6 Field Documentation . . . 633
15.156 smtrat::ElementWithOrigins< Element, Origin > Class Template Reference . . . 633
15.156.1 Constructor & Destructor Documentation . . . 633
15.156.2 Member Function Documentation . . . 634
15.157 smtrat::EMModule< Settings > Class Template Reference . . . 634
15.157.1 Member Typedef Documentation . . . 639
15.157.2 Member Enumeration Documentation . . . 639
15.157.3 Constructor & Destructor Documentation . . . 639
15.157.4 Member Function Documentation . . . 640

15.157.5 Field Documentation . . . 662
15.158 smtrat::EQGraph< VertexName > Struct Template Reference . . . 665
15.158.1 Member Typedef Documentation . . . 665
15.158.2 Member Function Documentation . . . 666
15.158.3 Field Documentation . . . 667
15.159 smtrat::EQModule< Settings > Class Template Reference . . . 667
15.159.1 Member Typedef Documentation . . . 672
15.159.2 Member Enumeration Documentation . . . 672
15.159.3 Constructor & Destructor Documentation . . . 672
15.159.4 Member Function Documentation . . . 673
15.159.5 Field Documentation . . . 694
15.160 smtrat::EQPreprocessingModule< Settings > Class Template Reference . . . 697
15.160.1 Member Typedef Documentation . . . 702
15.160.2 Member Enumeration Documentation . . . 702
15.160.3 Constructor & Destructor Documentation . . . 702
15.160.4 Member Function Documentation . . . 702
15.160.5 Field Documentation . . . 723
15.161 Minisat::Equal< K > Struct Template Reference . . . 726
15.161.1 Member Function Documentation . . . 726
15.162 delta::ErrorHandler Struct Reference . . . 726
15.162.1 Member Function Documentation . . . 726
15.163 smtrat::parser::ErrorHandler Struct Reference . . . 727
15.163.1 Member Function Documentation . . . 727
15.164 smtrat::ESModule< Settings > Class Template Reference . . . 727
15.164.1 Member Typedef Documentation . . . 732
15.164.2 Member Enumeration Documentation . . . 732
15.164.3 Constructor & Destructor Documentation . . . 732
15.164.4 Member Function Documentation . . . 733
15.164.5 Field Documentation . . . 755
15.165 smtrat::uf_impl::estimator< IteratorType, IsRandomAccess > Struct Template Reference . . . 758
15.165.1 Detailed Description . . . 758
15.165.2 Member Function Documentation . . . 758
15.166 smtrat::uf_impl::estimator< IteratorType, false > Struct Template Reference . . . 758
15.166.1 Member Function Documentation . . . 758
15.167 smtrat::ExactlyOneCommanderEncoder Class Reference . . . 758
15.167.1 Constructor & Destructor Documentation . . . 759
15.167.2 Member Function Documentation . . . 759
15.167.3 Field Documentation . . . 760
15.168 smtrat::execution::ExecutionState Class Reference . . . 760
15.168.1 Constructor & Destructor Documentation . . . 760
15.168.2 Member Function Documentation . . . 760
15.169 smtrat::Executor< Strategy > Class Template Reference . . . 763

15.169.1 Member Typedef Documentation 765
15.169.2 Constructor & Destructor Documentation 765
15.169.3 Member Function Documentation 765
15.169.4 Field Documentation 770
15.170 smtrat::mcsat::onecellcad::recursive::Explanation< Setting1, Setting2 > Struct Template Reference 770
15.170.1 Member Function Documentation 770
15.171 smtrat::mcsat::vs::Explanation Struct Reference 771
15.171.1 Member Function Documentation 771
15.172 smtrat::mcsat::fm::Explanation< Settings > Struct Template Reference 771
15.172.1 Member Function Documentation 771
15.173 smtrat::mcsat::icp::Explanation Struct Reference 771
15.173.1 Member Function Documentation 771
15.174 smtrat::mcsat::nlsat::Explanation Struct Reference 772
15.174.1 Member Function Documentation 772
15.175 smtrat::mcsat::onecell::Explanation Struct Reference 772
15.175.1 Member Function Documentation 772
15.176 smtrat::mcsat::onecellcad::levelwise::Explanation< Setting1, Setting2 > Struct Template Reference 773
15.176.1 Member Function Documentation 773
15.177 smtrat::mcsat::vs::ExplanationGenerator< Settings > Class Template Reference 773
15.177.1 Constructor & Destructor Documentation 773
15.177.2 Member Function Documentation 773
15.178 smtrat::mcsat::nlsat::ExplanationGenerator Class Reference 773
15.178.1 Constructor & Destructor Documentation 774
15.178.2 Member Function Documentation 774
15.179 smtrat::expression::Expression Class Reference 774
15.179.1 Constructor & Destructor Documentation 775
15.179.2 Member Function Documentation 775
15.179.3 Friends And Related Function Documentation 777
15.180 smtrat::expression::ExpressionContent Struct Reference 777
15.180.1 Member Typedef Documentation 777
15.180.2 Constructor & Destructor Documentation 777
15.180.3 Friends And Related Function Documentation 777
15.180.4 Field Documentation 778
15.181 smtrat::expression::ExpressionConverter Struct Reference 778
15.181.1 Member Function Documentation 778
15.181.2 Field Documentation 779
15.182 smtrat::expression::ExpressionModifier Class Reference 779
15.182.1 Member Typedef Documentation 779
15.182.2 Member Function Documentation 779
15.183 smtrat::expression::ExpressionPool Class Reference 780
15.183.1 Constructor & Destructor Documentation 781
15.183.2 Member Function Documentation 781

15.183.3 Friends And Related Function Documentation . . . 782
15.184 smtrat::parser::ParserState::ExpressionScope Struct Reference . . . 782
15.184.1 Constructor & Destructor Documentation . . . 782
15.184.2 Member Function Documentation . . . 782
15.185 smtrat::expression::ExpressionTypeChecker< T > Struct Template Reference . . . 782
15.185.1 Member Function Documentation . . . 782
15.186 smtrat::expression::ExpressionVisitor Class Reference . . . 782
15.186.1 Member Typedef Documentation . . . 783
15.186.2 Member Function Documentation . . . 783
15.187 smtrat::mcsat::FastParallelExplanation< Backends > Struct Template Reference . . . 784
15.187.1 Detailed Description . . . 784
15.188 smtrat::mcsat::variableordering::detail::FeatureCollector< Objects > Struct Template Reference . 784
15.188.1 Detailed Description . . . 784
15.188.2 Member Typedef Documentation . . . 784
15.188.3 Member Function Documentation . . . 785
15.188.4 Field Documentation . . . 785
15.189 smtrat::fixedsize_allocator< T > Class Template Reference . . . 785
15.189.1 Detailed Description . . . 785
15.189.2 Constructor & Destructor Documentation . . . 786
15.189.3 Member Function Documentation . . . 786
15.190 smtrat::fixedsize_allocator< void > Class Template Reference . . . 786
15.190.1 Detailed Description . . . 786
15.190.2 Field Documentation . . . 787
15.191 smtrat::impl::fixedsize_allocator_freelists Class Reference . . . 787
15.191.1 Field Documentation . . . 787
15.192 smtrat::impl::fixedsize_allocator_impl< T, SizeShift, LargeEnough, SmallEnough > Class Template Reference . . . 787
15.193 smtrat::impl::fixedsize_allocator_impl< T, SizeShift, true, true > Class Template Reference . . . 787
15.193.1 Member Typedef Documentation . . . 788
15.193.2 Member Function Documentation . . . 788
15.193.3 Field Documentation . . . 789
15.194 smtrat::impl::fixedsize_allocator_size_helper< T, Size > Class Template Reference . . . 789
15.195 smtrat::fixedsize_freelist< ChunkSizeShift > Class Template Reference . . . 789
15.195.1 Detailed Description . . . 790
15.195.2 Constructor & Destructor Documentation . . . 790
15.195.3 Member Function Documentation . . . 790
15.196 smtrat::impl::fixedsize_freelists< SizeCurrent, SizeMax, LowEnough > Class Template Reference 790
15.196.1 Detailed Description . . . 790
15.197 smtrat::impl::fixedsize_freelists< SizeCurrent, SizeMax, true > Class Template Reference . . . 791
15.198 smtrat::parser::FixedWidthConstant< T > Struct Template Reference . . . 791
15.198.1 Detailed Description . . . 791
15.198.2 Constructor & Destructor Documentation . . . 791

15.198.3 Member Function Documentation . . . 791
15.198.4 Field Documentation . . . 791
15.199 smtrat::parseformula::FormulaCollector Class Reference . . . 792
15.199.1 Member Typedef Documentation . . . 793
15.199.2 Member Function Documentation . . . 793
15.199.3 Field Documentation . . . 798
15.200 smtrat::ICPModule< Settings >::formulaPtrComp Struct Reference . . . 798
15.200.1 Member Function Documentation . . . 798
15.201 smtrat::FormulaWithOrigins Class Reference . . . 798
15.201.1 Detailed Description . . . 799
15.201.2 Constructor & Destructor Documentation . . . 799
15.201.3 Member Function Documentation . . . 800
15.201.4 Friends And Related Function Documentation . . . 800
15.202 smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices > Class Template Reference . . . 801
15.202.1 Detailed Description . . . 802
15.202.2 Member Typedef Documentation . . . 802
15.202.3 Member Function Documentation . . . 802
15.202.4 Friends And Related Function Documentation . . . 804
15.203 smtrat::FouMoModule< Settings > Class Template Reference . . . 804
15.203.1 Detailed Description . . . 809
15.203.2 Member Typedef Documentation . . . 810
15.203.3 Member Enumeration Documentation . . . 810
15.203.4 Constructor & Destructor Documentation . . . 810
15.203.5 Member Function Documentation . . . 810
15.203.6 Field Documentation . . . 831
15.204 smtrat::qe::fm::FourierMotzkinQE Class Reference . . . 834
15.204.1 Detailed Description . . . 834
15.204.2 Member Typedef Documentation . . . 834
15.204.3 Constructor & Destructor Documentation . . . 835
15.204.4 Member Function Documentation . . . 835
15.205 smtrat::FPPModule< Settings > Class Template Reference . . . 835
15.205.1 Member Typedef Documentation . . . 840
15.205.2 Member Enumeration Documentation . . . 840
15.205.3 Constructor & Destructor Documentation . . . 840
15.205.4 Member Function Documentation . . . 840
15.205.5 Field Documentation . . . 862
15.206 smtrat::freelist< T > Class Template Reference . . . 865
15.206.1 Detailed Description . . . 865
15.206.2 Constructor & Destructor Documentation . . . 866
15.206.3 Member Function Documentation . . . 866
15.207 smtrat::mcsat::FullParallelExplanation< Backends > Struct Template Reference . . . 867

15.207.1 Detailed Description . . . 867
15.208 smtrat::cad::FullSampleComparator< Iterator, Strategy > Struct Template Reference . . . 867
15.209 smtrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::T > Struct Template Reference . . . 867
15.210 smtrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::Type > Struct Template Reference . . . 867
15.211 smtrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::Value > Struct Template Reference . . . 867
15.212 smtrat::parser::FunctionInstantiator Struct Reference . . . 867
15.212.1 Constructor & Destructor Documentation . . . 867
15.212.2 Member Function Documentation . . . 868
15.213 smtrat::GBModule< Settings > Class Template Reference . . . 868
15.213.1 Detailed Description . . . 874
15.213.2 Member Typedef Documentation . . . 875
15.213.3 Member Enumeration Documentation . . . 875
15.213.4 Constructor & Destructor Documentation . . . 875
15.213.5 Member Function Documentation . . . 875
15.213.6 Friends And Related Function Documentation . . . 902
15.213.7 Field Documentation . . . 902
15.214 smtrat::GBModuleState< Settings > Class Template Reference . . . 906
15.214.1 Detailed Description . . . 906
15.214.2 Constructor & Destructor Documentation . . . 906
15.214.3 Member Function Documentation . . . 906
15.214.4 Field Documentation . . . 906
15.215 smtrat::GBPPModule< Settings > Class Template Reference . . . 907
15.215.1 Member Typedef Documentation . . . 912
15.215.2 Member Enumeration Documentation . . . 912
15.215.3 Constructor & Destructor Documentation . . . 912
15.215.4 Member Function Documentation . . . 912
15.215.5 Field Documentation . . . 935
15.216 smtrat::EQModule< Settings >::not_asserted_equality::hash Struct Reference . . . 938
15.216.1 Member Function Documentation . . . 938
15.217 Minisat::Hash< K > Struct Template Reference . . . 938
15.217.1 Member Function Documentation . . . 938
15.218 std::hash< const smtrat::expression::ExpressionContent * > Struct Template Reference . . . 938
15.218.1 Member Function Documentation . . . 938
15.219 std::hash< smtrat::expression::BinaryExpression > Struct Template Reference . . . 938
15.219.1 Member Function Documentation . . . 938
15.220 std::hash< smtrat::expression::Expression > Struct Template Reference . . . 939
15.220.1 Member Function Documentation . . . 939
15.221 std::hash< smtrat::expression::ITEExpression > Struct Template Reference . . . 939
15.221.1 Member Function Documentation . . . 939
15.222 std::hash< smtrat::expression::NaryExpression > Struct Template Reference . . . 939

15.222.1 Member Function Documentation . . . 939
15.223 std::hash< smtrat::expression::QuantifierExpression > Struct Template Reference . . . 939
15.223.1 Member Function Documentation . . . 939
15.224 std::hash< smtrat::expression::UnaryExpression > Struct Template Reference . . . 939
15.224.1 Member Function Documentation . . . 940
15.225 std::hash< smtrat::lra::Variable< T1, T2 > > Struct Template Reference . . . 940
15.225.1 Detailed Description . . . 940
15.225.2 Member Function Documentation . . . 940
15.226 std::hash< smtrat::vs::Substitution > Struct Template Reference . . . 940
15.226.1 Member Function Documentation . . . 940
15.227 std::hash< std::vector< T > > Struct Template Reference . . . 940
15.227.1 Member Function Documentation . . . 941
15.228 std::hash_combiner Struct Reference . . . 941
15.228.1 Constructor & Destructor Documentation . . . 941
15.228.2 Member Function Documentation . . . 941
15.228.3 Field Documentation . . . 941
15.229 Minisat::Heap< Comp > Class Template Reference . . . 941
15.229.1 Constructor & Destructor Documentation . . . 942
15.229.2 Member Function Documentation . . . 942
15.230 smtrat::parser::HexadecimalParser Struct Reference . . . 943
15.230.1 Detailed Description . . . 943
15.230.2 Member Typedef Documentation . . . 943
15.230.3 Constructor & Destructor Documentation . . . 943
15.230.4 Member Function Documentation . . . 944
15.230.5 Field Documentation . . . 944
15.231 smtrat::icp::HistoryNode Class Reference . . . 944
15.231.1 Constructor & Destructor Documentation . . . 945
15.231.2 Member Function Documentation . . . 945
15.232 smtrat::ICEModule< Settings > Class Template Reference . . . 948
15.232.1 Member Typedef Documentation . . . 953
15.232.2 Member Enumeration Documentation . . . 953
15.232.3 Constructor & Destructor Documentation . . . 953
15.232.4 Member Function Documentation . . . 954
15.232.5 Field Documentation . . . 975
15.233 smtrat::ICPModule< Settings > Class Template Reference . . . 978
15.233.1 Member Typedef Documentation . . . 984
15.233.2 Member Enumeration Documentation . . . 984
15.233.3 Constructor & Destructor Documentation . . . 984
15.233.4 Member Function Documentation . . . 984
15.233.5 Field Documentation . . . 1006
15.234 smtrat::icp::IcpVariable Class Reference . . . 1009
15.234.1 Constructor & Destructor Documentation . . . 1009

15.234.2 Member Function Documentation . . . 1009
15.234.3 Friends And Related Function Documentation . . . 1012
15.235 smtrat::icp::icpVariableComp Struct Reference . . . 1012
15.235.1 Member Function Documentation . . . 1012
15.236 smtrat::parser::Identifier Struct Reference . . . 1012
15.236.1 Constructor & Destructor Documentation . . . 1013
15.236.2 Member Function Documentation . . . 1013
15.236.3 Field Documentation . . . 1013
15.237 smtrat::parser::IdentifierParser Struct Reference . . . 1014
15.237.1 Constructor & Destructor Documentation . . . 1014
15.237.2 Field Documentation . . . 1014
15.238 smtrat::cad::debug::IDSanitizer Struct Reference . . . 1014
15.238.1 Constructor & Destructor Documentation . . . 1014
15.238.2 Member Function Documentation . . . 1014
15.239 smtrat::mcsat::fm::IgnoreCoreSettings Struct Reference . . . 1014
15.239.1 Field Documentation . . . 1015
15.240 smtrat::IncWidthModule< Settings > Class Template Reference . . . 1015
15.240.1 Member Typedef Documentation . . . 1020
15.240.2 Member Enumeration Documentation . . . 1020
15.240.3 Constructor & Destructor Documentation . . . 1020
15.240.4 Member Function Documentation . . . 1020
15.240.5 Field Documentation . . . 1041
15.241 smtrat::mcsat::onecellcad::firstlevel::indexed_root Struct Reference . . . 1044
15.241.1 Field Documentation . . . 1044
15.242 smtrat::parser::IndexedFunctionInstantiator Struct Reference . . . 1044
15.242.1 Constructor & Destructor Documentation . . . 1045
15.242.2 Member Function Documentation . . . 1045
15.243 smtrat::cadcells::datastructures::IndexedRoot Struct Reference . . . 1045
15.243.1 Detailed Description . . . 1045
15.243.2 Constructor & Destructor Documentation . . . 1045
15.243.3 Field Documentation . . . 1045
15.244 smtrat::cadcells::datastructures::IndexedRootOrdering Class Reference . . . 1046
15.244.1 Detailed Description . . . 1046
15.244.2 Member Function Documentation . . . 1046
15.245 smtrat::InequalitiesTable< Settings > Class Template Reference . . . 1047
15.245.1 Detailed Description . . . 1048
15.245.2 Member Typedef Documentation . . . 1048
15.245.3 Constructor & Destructor Documentation . . . 1049
15.245.4 Member Function Documentation . . . 1049
15.245.5 Field Documentation . . . 1050
15.246 smtrat::lra::Bound< T1, T2 >::Info Struct Reference . . . 1050
15.246.1 Detailed Description . . . 1051

15.246.2 Constructor & Destructor Documentation . . . 1051
15.246.3 Field Documentation . . . 1051
15.247 smtrat::mcsat::InformationGetter Struct Reference . . . 1052
15.247.1 Field Documentation . . . 1052
15.248 smtrat::parser::Instantiator< V, T > Struct Template Reference . . . 1053
15.248.1 Member Function Documentation . . . 1054
15.248.2 Field Documentation . . . 1055
15.249 smtrat::parser::InstructionHandler Class Reference . . . 1055
15.249.1 Member Typedef Documentation . . . 1056
15.249.2 Constructor & Destructor Documentation . . . 1056
15.249.3 Member Function Documentation . . . 1056
15.249.4 Field Documentation . . . 1061
15.250 Minisat::Int64Range Struct Reference . . . 1061
15.250.1 Constructor & Destructor Documentation . . . 1062
15.250.2 Field Documentation . . . 1062
15.251 smtrat::IntBlastModule< Settings > Class Template Reference . . . 1062
15.251.1 Member Typedef Documentation . . . 1067
15.251.2 Member Enumeration Documentation . . . 1067
15.251.3 Constructor & Destructor Documentation . . . 1067
15.251.4 Member Function Documentation . . . 1067
15.251.5 Field Documentation . . . 1088
15.252 smtrat::IntEqModule< Settings > Class Template Reference . . . 1091
15.252.1 Detailed Description . . . 1096
15.252.2 Member Typedef Documentation . . . 1096
15.252.3 Member Enumeration Documentation . . . 1096
15.252.4 Constructor & Destructor Documentation . . . 1096
15.252.5 Member Function Documentation . . . 1096
15.252.6 Field Documentation . . . 1117
15.253 smtrat::mcsat::icp::IntervalPropagation Class Reference . . . 1120
15.253.1 Constructor & Destructor Documentation . . . 1120
15.253.2 Member Function Documentation . . . 1120
15.254 Minisat::IntOption Class Reference . . . 1121
15.254.1 Constructor & Destructor Documentation . . . 1121
15.254.2 Member Function Documentation . . . 1121
15.254.3 Field Documentation . . . 1122
15.255 Minisat::IntRange Struct Reference . . . 1122
15.255.1 Constructor & Destructor Documentation . . . 1122
15.255.2 Field Documentation . . . 1122
15.256 smtrat::is_sample_outside< S > Struct Template Reference . . . 1123
15.256.1 Member Function Documentation . . . 1123
15.257 smtrat::is_sample_outside< IsSampleOutsideAlgorithm::DEFAULT > Struct Template Reference . 1123
15.257.1 Member Function Documentation . . . 1123

15.258 smtrat::is_variant< T > Struct Template Reference . . . 1123
15.258.1 Detailed Description . . . 1123
15.259 smtrat::expression::ITEExpression Struct Reference . . . 1123
15.259.1 Constructor & Destructor Documentation . . . 1124
15.259.2 Field Documentation . . . 1124
15.260 benchmax::RandomizationAdaptor< T >::iterator Struct Reference . . . 1124
15.260.1 Constructor & Destructor Documentation . . . 1124
15.260.2 Member Function Documentation . . . 1125
15.261 smtrat::ModuleInput::IteratorCompare Struct Reference . . . 1125
15.261.1 Member Function Documentation . . . 1125
15.262 benchmax::Jobs Class Reference . . . 1125
15.262.1 Detailed Description . . . 1126
15.262.2 Constructor & Destructor Documentation . . . 1126
15.262.3 Member Function Documentation . . . 1126
15.263 smtrat::JunctorMerger Struct Reference . . . 1126
15.263.1 Member Function Documentation . . . 1126
15.264 smtrat::parser::KeywordParser Struct Reference . . . 1127
15.264.1 Detailed Description . . . 1127
15.264.2 Constructor & Destructor Documentation . . . 1127
15.264.3 Field Documentation . . . 1127
15.265 Minisat::lbool Class Reference . . . 1127
15.265.1 Constructor & Destructor Documentation . . . 1128
15.265.2 Member Function Documentation . . . 1128
15.265.3 Friends And Related Function Documentation . . . 1128
15.266 smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound Struct Reference . . . 1128
15.266.1 Constructor & Destructor Documentation . . . 1129
15.266.2 Field Documentation . . . 1129
15.267 smtrat::Module::Lemma Struct Reference . . . 1130
15.267.1 Constructor & Destructor Documentation . . . 1130
15.267.2 Field Documentation . . . 1130
15.268 Minisat::LessThan_default< T > Struct Template Reference . . . 1131
15.268.1 Member Function Documentation . . . 1131
15.269 smtrat::cad::sample_compare::level Struct Reference . . . 1131
15.270 smtrat::cad::projection_compare::level Struct Reference . . . 1131
15.271 smtrat::cad::ProjectionLevelInformation::LevelInfo Struct Reference . . . 1131
15.271.1 Member Function Documentation . . . 1131
15.271.2 Field Documentation . . . 1132
15.272 smtrat::mcsat::onecellcad::levelwise::LevelwiseCAD Class Reference . . . 1133
15.272.1 Member Function Documentation . . . 1133
15.272.2 Field Documentation . . . 1135
15.273 smtrat::LevelWiseInformation< Settings > Class Template Reference . . . 1135
15.273.1 Constructor & Destructor Documentation . . . 1136

15.273.2 Member Function Documentation . . . 1136
15.274 smtrat::cad::LiftingTree< Settings > Class Template Reference . . . 1138
15.274.1 Member Typedef Documentation . . . 1138
15.274.2 Constructor & Destructor Documentation . . . 1139
15.274.3 Member Function Documentation . . . 1139
15.275 smtrat::resource::Limiter Class Reference . . . 1141
15.275.1 Member Function Documentation . . . 1141
15.276 smtrat::ICPModule< Settings >::linearVariable Struct Reference . . . 1142
15.276.1 Field Documentation . . . 1142
15.277 smtrat::mcsat::ClauseChain::Link Struct Reference . . . 1142
15.277.1 Constructor & Destructor Documentation . . . 1142
15.277.2 Member Function Documentation . . . 1142
15.277.3 Friends And Related Function Documentation . . . 1143
15.278 Minisat::Lit Struct Reference . . . 1143
15.278.1 Member Function Documentation . . . 1143
15.278.2 Friends And Related Function Documentation . . . 1143
15.278.3 Field Documentation . . . 1143
15.279 benchmax::LocalBackend Class Reference . . . 1143
15.279.1 Detailed Description . . . 1144
15.279.2 Member Function Documentation . . . 1144
15.279.3 Field Documentation . . . 1145
15.280 smtrat::LOG Class Reference . . . 1146
15.280.1 Member Function Documentation . . . 1146
15.280.2 Friends And Related Function Documentation . . . 1146
15.280.3 Field Documentation . . . 1146
15.281 smtrat::parser::LogicParser Struct Reference . . . 1146
15.281.1 Constructor & Destructor Documentation . . . 1146
15.282 smtrat::LongFormulaEncoder Class Reference . . . 1146
15.282.1 Constructor & Destructor Documentation . . . 1147
15.282.2 Member Function Documentation . . . 1147
15.282.3 Field Documentation . . . 1147
15.283 smtrat::LRAModule< Settings > Class Template Reference . . . 1148
15.283.1 Detailed Description . . . 1154
15.283.2 Member Typedef Documentation . . . 1154
15.283.3 Member Enumeration Documentation . . . 1155
15.283.4 Constructor & Destructor Documentation . . . 1156
15.283.5 Member Function Documentation . . . 1156
15.283.6 Field Documentation . . . 1180
15.284 smtrat::LVEModule< Settings > Class Template Reference . . . 1183
15.284.1 Member Typedef Documentation . . . 1188
15.284.2 Member Enumeration Documentation . . . 1188
15.284.3 Constructor & Destructor Documentation . . . 1189

15.284.4 Member Function Documentation . . . 1189
15.284.5 Field Documentation . . . 1210
15.285 smtrat::Manager Class Reference . . . 1213
15.285.1 Detailed Description . . . 1215
15.285.2 Constructor & Destructor Documentation . . . 1215
15.285.3 Member Function Documentation . . . 1215
15.285.4 Friends And Related Function Documentation . . . 1223
15.286 Minisat::Map< K, D, H, E > Class Template Reference . . . 1223
15.286.1 Constructor & Destructor Documentation . . . 1224
15.286.2 Member Function Documentation . . . 1224
15.287 benchmax::MathSAT Class Reference . . . 1225
15.287.1 Detailed Description . . . 1226
15.287.2 Constructor & Destructor Documentation . . . 1226
15.287.3 Member Function Documentation . . . 1226
15.287.4 Field Documentation . . . 1228
15.288 smtrat::mcsat::fm::MaxSizeComparator Struct Reference . . . 1228
15.288.1 Detailed Description . . . 1228
15.288.2 Member Function Documentation . . . 1228
15.288.3 Field Documentation . . . 1228
15.289 smtrat::MaxSMT< Solver, Strategy > Class Template Reference . . . 1228
15.289.1 Constructor & Destructor Documentation . . . 1229
15.289.2 Member Function Documentation . . . 1229
15.290 smtrat::maxsmt::MaxSMTBackend< Solver, Strategy > Class Template Reference . . . 1229
15.291 smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::FU_MALIK_INCREMENTAL > Class Template Reference . . . 1229
15.291.1 Constructor & Destructor Documentation . . . 1230
15.291.2 Member Function Documentation . . . 1230
15.292 smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::LINEAR_SEARCH > Class Template Reference . . . 1230
15.292.1 Constructor & Destructor Documentation . . . 1230
15.292.2 Member Function Documentation . . . 1230
15.293 smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::MSU3 > Class Template Reference . . . 1230
15.293.1 Constructor & Destructor Documentation . . . 1230
15.293.2 Member Function Documentation . . . 1230
15.294 smtrat::MCBModule< Settings > Class Template Reference . . . 1230
15.294.1 Member Typedef Documentation . . . 1236
15.294.2 Member Enumeration Documentation . . . 1236
15.294.3 Constructor & Destructor Documentation . . . 1236
15.294.4 Member Function Documentation . . . 1236
15.294.5 Field Documentation . . . 1259
15.295 smtrat::mcsat::MCSATBackend< Settings > Class Template Reference . . . 1262
15.295.1 Member Function Documentation . . . 1262
15.296 smtrat::mcsat::MCSATMixin< Settings > Class Template Reference . . . 1263

15.296.1 Constructor & Destructor Documentation . . . 1265
15.296.2 Member Function Documentation . . . 1265
15.296.3 Friends And Related Function Documentation . . . 1269
15.297 smtrat::expression::simplifier::MergeSimplifier Struct Reference . . . 1270
15.297.1 Member Function Documentation . . . 1270
15.298 benchmax::Minisat Class Reference . . . 1271
15.298.1 Detailed Description . . . 1272
15.298.2 Constructor & Destructor Documentation . . . 1272
15.298.3 Member Function Documentation . . . 1272
15.298.4 Field Documentation . . . 1274
15.299 benchmax::Minisatp Class Reference . . . 1274
15.299.1 Detailed Description . . . 1275
15.299.2 Constructor & Destructor Documentation . . . 1275
15.299.3 Member Function Documentation . . . 1275
15.299.4 Field Documentation . . . 1276
15.300 smtrat::mcsat::fm::MinSizeComparator Struct Reference . . . 1277
15.300.1 Detailed Description . . . 1277
15.300.2 Member Function Documentation . . . 1277
15.300.3 Field Documentation . . . 1277
15.301 smtrat::mcsat::fm::MinVarCountComparator Struct Reference . . . 1277
15.301.1 Detailed Description . . . 1277
15.301.2 Member Function Documentation . . . 1277
15.301.3 Field Documentation . . . 1277
15.302 smtrat::cad::MISGeneration< heuristic > Class Template Reference . . . 1277
15.302.1 Member Function Documentation . . . 1278
15.303 smtrat::MixedSignEncoder Class Reference . . . 1279
15.303.1 Constructor & Destructor Documentation . . . 1279
15.303.2 Member Function Documentation . . . 1279
15.303.3 Field Documentation . . . 1280
15.304 smtrat::cad::ModelBasedProjection< incrementality, backtracking, Settings > Class Template Reference . . . 1280
15.304.1 Member Typedef Documentation . . . 1281
15.304.2 Member Function Documentation . . . 1281
15.304.3 Field Documentation . . . 1285
15.305 smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings > Class Template Reference . . . 1286
15.305.1 Detailed Description . . . 1287
15.305.2 Constructor & Destructor Documentation . . . 1287
15.305.3 Member Function Documentation . . . 1288
15.305.4 Friends And Related Function Documentation . . . 1292
15.305.5 Field Documentation . . . 1292
15.306 smtrat::Module Class Reference . . . 1293
15.306.1 Detailed Description . . . 1298

15.306.2 Member Enumeration Documentation . . . 1298
15.306.3 Constructor & Destructor Documentation . . . 1298
15.306.4 Member Function Documentation . . . 1299
15.306.5 Field Documentation . . . 1320
15.307 smtrat::ModuleFactory< Module > Struct Template Reference . . . 1323
15.307.1 Constructor & Destructor Documentation . . . 1323
15.307.2 Member Function Documentation . . . 1323
15.308 smtrat::ModuleInput Class Reference . . . 1324
15.308.1 Detailed Description . . . 1325
15.308.2 Member Typedef Documentation . . . 1325
15.308.3 Constructor & Destructor Documentation . . . 1325
15.308.4 Member Function Documentation . . . 1325
15.308.5 Friends And Related Function Documentation . . . 1330
15.309 smtrat::settings::ModuleSettings Struct Reference . . . 1330
15.309.1 Constructor & Destructor Documentation . . . 1331
15.309.2 Member Function Documentation . . . 1331
15.309.3 Field Documentation . . . 1331
15.310 smtrat::Module::ModuleStatistics Struct Reference . . . 1331
15.310.1 Member Function Documentation . . . 1331
15.311 smtrat::ModuleWrapper< M > Class Template Reference . . . 1332
15.311.1 Constructor & Destructor Documentation . . . 1332
15.311.2 Member Function Documentation . . . 1332
15.312 smtrat::MonomialMappingByVariablePool Class Reference . . . 1333
15.312.1 Member Function Documentation . . . 1333
15.312.2 Friends And Related Function Documentation . . . 1333
15.313 smtrat::expression::NaryExpression Struct Reference . . . 1333
15.313.1 Constructor & Destructor Documentation . . . 1334
15.313.2 Member Function Documentation . . . 1334
15.313.3 Field Documentation . . . 1334
15.314 smtrat::expression::simplifier::NegationSimplifier Struct Reference . . . 1334
15.314.1 Member Function Documentation . . . 1334
15.315 smtrat::NewCADModule< Settings > Class Template Reference . . . 1336
15.315.1 Member Typedef Documentation . . . 1341
15.315.2 Member Enumeration Documentation . . . 1341
15.315.3 Constructor & Destructor Documentation . . . 1341
15.315.4 Member Function Documentation . . . 1341
15.315.5 Field Documentation . . . 1363
15.316 smtrat::NewCoveringModule< Settings > Class Template Reference . . . 1366
15.316.1 Member Typedef Documentation . . . 1371
15.316.2 Member Enumeration Documentation . . . 1371
15.316.3 Constructor & Destructor Documentation . . . 1371
15.316.4 Member Function Documentation . . . 1371

15.316.5 Field Documentation . . . 1392
15.317 smtrat::NNFPreparation Struct Reference . . . 1395
15.317.1 Member Function Documentation . . . 1395
15.318 smtrat::NNFRewriter Class Reference . . . 1396
15.318.1 Constructor & Destructor Documentation . . . 1396
15.318.2 Member Function Documentation . . . 1396
15.319 benchmax::ssh::Node Struct Reference . . . 1397
15.319.1 Detailed Description . . . 1397
15.319.2 Field Documentation . . . 1397
15.320 delta::Node Struct Reference . . . 1398
15.320.1 Detailed Description . . . 1399
15.320.2 Constructor & Destructor Documentation . . . 1399
15.320.3 Member Function Documentation . . . 1400
15.320.4 Field Documentation . . . 1401
15.321 delta::NodePrinter< pretty > Struct Template Reference . . . 1402
15.321.1 Constructor & Destructor Documentation . . . 1402
15.321.2 Member Function Documentation . . . 1402
15.321.3 Field Documentation . . . 1402
15.322 smtrat::mcsat::onecellcad::recursive::NoHeuristic Struct Reference . . . 1403
15.322.1 Field Documentation . . . 1403
15.323 smtrat::NRAILModule< Settings > Class Template Reference . . . 1403
15.323.1 Member Typedef Documentation . . . 1408
15.323.2 Member Enumeration Documentation . . . 1408
15.323.3 Constructor & Destructor Documentation . . . 1408
15.323.4 Member Function Documentation . . . 1408
15.323.5 Field Documentation . . . 1430
15.324 smtrat::parser::NumeralParser Struct Reference . . . 1433
15.324.1 Detailed Description . . . 1433
15.325 smtrat::lra::Numeric Class Reference . . . 1433
15.325.1 Constructor & Destructor Documentation . . . 1434
15.325.2 Member Function Documentation . . . 1435
15.326 smtrat::execution::Objective Struct Reference . . . 1440
15.326.1 Field Documentation . . . 1440
15.327 Minisat::OccLists< Idx, Vec, Deleted > Class Template Reference . . . 1440
15.327.1 Constructor & Destructor Documentation . . . 1440
15.327.2 Member Function Documentation . . . 1440
15.328 smtrat::mcsat::onecellcad::OneCellCAD Class Reference . . . 1441
15.328.1 Constructor & Destructor Documentation . . . 1442
15.328.2 Member Function Documentation . . . 1442
15.328.3 Field Documentation . . . 1443
15.329 benchmax::settings::OperationSettings Struct Reference . . . 1443
15.329.1 Detailed Description . . . 1443

15.329.2 Field Documentation . . . 1443
15.330 smtrat::Optimization< Solver > Class Template Reference . . . 1444
15.330.1 Constructor & Destructor Documentation . . . 1444
15.330.2 Member Function Documentation . . . 1444
15.331 Minisat::Option Class Reference . . . 1445
15.331.1 Constructor & Destructor Documentation . . . 1445
15.331.2 Member Function Documentation . . . 1445
15.331.3 Friends And Related Function Documentation . . . 1446
15.331.4 Field Documentation . . . 1446
15.332 Minisat::Option::OptionLt Struct Reference . . . 1446
15.332.1 Member Function Documentation . . . 1446
15.333 smtrat::datastructures::OrderPair< T1, T2, reversed > Struct Template Reference . . . 1447
15.333.1 Constructor & Destructor Documentation . . . 1447
15.333.2 Member Function Documentation . . . 1447
15.334 smtrat::datastructures::OrderPair< T1, T2, true > Struct Template Reference . . . 1447
15.334.1 Constructor & Destructor Documentation . . . 1447
15.334.2 Member Function Documentation . . . 1447
15.335 smtrat::cad::Origin Class Reference . . . 1447
15.335.1 Detailed Description . . . 1448
15.335.2 Constructor & Destructor Documentation . . . 1448
15.335.3 Member Function Documentation . . . 1449
15.335.4 Friends And Related Function Documentation . . . 1450
15.336 smtrat::cad::preprocessor::Origins Struct Reference . . . 1450
15.336.1 Member Function Documentation . . . 1450
15.336.2 Field Documentation . . . 1451
15.337 smtrat::OrPathShortener Struct Reference . . . 1451
15.337.1 Detailed Description . . . 1451
15.337.2 Constructor & Destructor Documentation . . . 1451
15.337.3 Member Function Documentation . . . 1451
15.338 Minisat::OutOfMemoryException Class Reference . . . 1452
15.339 smtrat::parser::OutputWrapper Class Reference . . . 1452
15.339.1 Constructor & Destructor Documentation . . . 1452
15.339.2 Member Function Documentation . . . 1452
15.340 Minisat::Map< K, D, H, E >::Pair Struct Reference . . . 1452
15.340.1 Field Documentation . . . 1452
15.341 smtrat::pairhash< F, S, FirstHasher, SecondHasher > Struct Template Reference . . . 1453
15.341.1 Detailed Description . . . 1453
15.341.2 Member Function Documentation . . . 1453
15.342 smtrat::mcsat::ParallelExplanation< Backends > Struct Template Reference . . . 1453
15.342.1 Member Function Documentation . . . 1453
15.343 delta::Parser Class Reference . . . 1453
15.343.1 Detailed Description . . . 1454

15.343.2 Constructor & Destructor Documentation . . . 1454
15.343.3 Member Function Documentation . . . 1454
15.344 smtrat::parser::ParserSettings Struct Reference . . . 1454
15.344.1 Field Documentation . . . 1454
15.345 smtrat::parser::ParserState Struct Reference . . . 1455
15.345.1 Constructor & Destructor Documentation . . . 1456
15.345.2 Member Function Documentation . . . 1456
15.345.3 Field Documentation . . . 1457
15.346 smtrat::PBGaussModule< Settings > Class Template Reference . . . 1458
15.346.1 Member Typedef Documentation . . . 1464
15.346.2 Member Enumeration Documentation . . . 1464
15.346.3 Constructor & Destructor Documentation . . . 1464
15.346.4 Member Function Documentation . . . 1464
15.346.5 Field Documentation . . . 1487
15.347 smtrat::PBPPModule< Settings > Class Template Reference . . . 1490
15.347.1 Member Typedef Documentation . . . 1495
15.347.2 Member Enumeration Documentation . . . 1495
15.347.3 Constructor & Destructor Documentation . . . 1495
15.347.4 Member Function Documentation . . . 1495
15.347.5 Field Documentation . . . 1516
15.348 smtrat::PersistentUnionFind Struct Reference . . . 1519
15.348.1 Member Typedef Documentation . . . 1519
15.348.2 Member Function Documentation . . . 1519
15.348.3 Field Documentation . . . 1520
15.349 smtrat::PFEModule< Settings > Class Template Reference . . . 1520
15.349.1 Member Typedef Documentation . . . 1526
15.349.2 Member Enumeration Documentation . . . 1526
15.349.3 Constructor & Destructor Documentation . . . 1526
15.349.4 Member Function Documentation . . . 1526
15.349.5 Field Documentation . . . 1547
15.350 smtrat::PModule Class Reference . . . 1550
15.350.1 Member Enumeration Documentation . . . 1555
15.350.2 Constructor & Destructor Documentation . . . 1556
15.350.3 Member Function Documentation . . . 1556
15.350.4 Field Documentation . . . 1578
15.351 smtrat::cadcells::operators::properties::poly_irreducible_sgn_inv Struct Reference . . . 1581
15.351.1 Member Function Documentation . . . 1581
15.351.2 Field Documentation . . . 1581
15.352 smtrat::cadcells::operators::properties::poly_ord_inv Struct Reference . . . 1581
15.352.1 Member Function Documentation . . . 1581
15.352.2 Field Documentation . . . 1582
15.353 smtrat::cadcells::operators::properties::poly_pdel Struct Reference . . . 1582

15.353.1 Member Function Documentation . . . 1582
15.353.2 Field Documentation . . . 1582
15.354 smtrat::cadcells::operators::properties::poly_sgn_inv Struct Reference . . . 1582
15.354.1 Member Function Documentation . . . 1583
15.354.2 Field Documentation . . . 1583
15.355 smtrat::cad::ProjectionPolynomialInformation::PolyInfo Struct Reference . . . 1583
15.355.1 Field Documentation . . . 1583
15.356 smtrat::cad::PolynomialComparator< PolynomialGetter > Struct Template Reference . . . 1583
15.356.1 Constructor & Destructor Documentation . . . 1583
15.356.2 Member Function Documentation . . . 1583
15.357 smtrat::cad::PolynomialLiftingQueue< PolynomialGetter > Class Template Reference . . . 1584
15.357.1 Constructor & Destructor Documentation . . . 1584
15.357.2 Member Function Documentation . . . 1584
15.357.3 Friends And Related Function Documentation . . . 1585
15.358 smtrat::cadcells::datastructures::PolyPool Class Reference . . . 1585
15.358.1 Detailed Description . . . 1585
15.358.2 Constructor & Destructor Documentation . . . 1585
15.358.3 Member Function Documentation . . . 1585
15.359 smtrat::cadcells::datastructures::detail::PolyProperties Struct Reference . . . 1586
15.359.1 Field Documentation . . . 1586
15.360 smtrat::cadcells::datastructures::PolyRef Struct Reference . . . 1587
15.360.1 Detailed Description . . . 1587
15.360.2 Field Documentation . . . 1587
15.361 smtrat::PolyTree Class Reference . . . 1587
15.361.1 Member Enumeration Documentation . . . 1587
15.361.2 Constructor & Destructor Documentation . . . 1587
15.361.3 Member Function Documentation . . . 1588
15.362 smtrat::PolyTreeContent Class Reference . . . 1588
15.362.1 Constructor & Destructor Documentation . . . 1588
15.362.2 Member Function Documentation . . . 1589
15.362.3 Friends And Related Function Documentation . . . 1589
15.362.4 Field Documentation . . . 1589
15.363 smtrat::PolyTreePool Class Reference . . . 1589
15.363.1 Constructor & Destructor Documentation . . . 1589
15.363.2 Member Function Documentation . . . 1589
15.364 smtrat::cad::Preprocessor Class Reference . . . 1589
15.364.1 Constructor & Destructor Documentation . . . 1590
15.364.2 Member Function Documentation . . . 1590
15.364.3 Friends And Related Function Documentation . . . 1591
15.365 smtrat::cad::PreprocessorSettings Struct Reference . . . 1591
15.365.1 Member Function Documentation . . . 1591
15.365.2 Field Documentation . . . 1591

15.366 benchmax::settings::PresetSettings Struct Reference . . . 1591
15.366.1 Field Documentation . . . 1591
15.367 smtrat::PriorityQueue< T, Compare > Class Template Reference . . . 1592
15.367.1 Constructor & Destructor Documentation . . . 1592
15.367.2 Member Function Documentation . . . 1592
15.367.3 Field Documentation . . . 1593
15.368 delta::Producer Class Reference . . . 1594
15.368.1 Detailed Description . . . 1594
15.368.2 Constructor & Destructor Documentation . . . 1594
15.368.3 Member Function Documentation . . . 1594
15.369 delta::ProgressBar Class Reference . . . 1594
15.369.1 Detailed Description . . . 1595
15.369.2 Member Function Documentation . . . 1595
15.370 smtrat::cad::Projection< incrementality, backtracking, Settings > Class Template Reference . . . 1596
15.370.1 Member Typedef Documentation . . . 1597
15.370.2 Member Function Documentation . . . 1597
15.370.3 Field Documentation . . . 1601
15.371 smtrat::qe::cad::Projection< Settings > Class Template Reference . . . 1601
15.371.1 Constructor & Destructor Documentation . . . 1603
15.371.2 Member Function Documentation . . . 1603
15.371.3 Friends And Related Function Documentation . . . 1607
15.371.4 Field Documentation . . . 1608
15.372 smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings > Class Template Reference . . . 1608
15.372.1 Constructor & Destructor Documentation . . . 1609
15.372.2 Member Function Documentation . . . 1609
15.372.3 Friends And Related Function Documentation . . . 1614
15.372.4 Field Documentation . . . 1614
15.373 smtrat::cad::Projection< Incrementality::FULL, BT, Settings > Class Template Reference . . . 1614
15.373.1 Constructor & Destructor Documentation . . . 1615
15.373.2 Member Function Documentation . . . 1615
15.373.3 Friends And Related Function Documentation . . . 1619
15.373.4 Field Documentation . . . 1619
15.374 smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings > Class Template Reference . . . 1620
15.374.1 Detailed Description . . . 1621
15.374.2 Constructor & Destructor Documentation . . . 1622
15.374.3 Member Function Documentation . . . 1622
15.374.4 Friends And Related Function Documentation . . . 1626
15.374.5 Field Documentation . . . 1626
15.375 smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings > Class Template Reference . . . 1627
15.375.1 Detailed Description . . . 1628

15.375.2 Constructor & Destructor Documentation . . . 1628
15.375.3 Member Function Documentation . . . 1628
15.375.4 Friends And Related Function Documentation . . . 1633
15.375.5 Field Documentation . . . 1633
15.376 smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings > Class Template Reference . . . 1633
15.376.1 Constructor & Destructor Documentation . . . 1634
15.376.2 Member Function Documentation . . . 1635
15.376.3 Friends And Related Function Documentation . . . 1639
15.376.4 Field Documentation . . . 1639
15.377 smtrat::cad::projection_compare::ProjectionComparator< Strategy > Struct Template Reference . 1639
15.378 smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::D > Struct Template Reference . . . 1639
15.378.1 Member Function Documentation . . . 1639
15.379 smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::ID > Struct Template Reference . . . 1639
15.379.1 Member Function Documentation . . . 1640
15.380 smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::LD > Struct Template Reference . . . 1640
15.380.1 Member Function Documentation . . . 1640
15.381 smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::PD > Struct Template Reference . . . 1640
15.381.1 Member Function Documentation . . . 1640
15.382 smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::SD > Struct Template Reference . . . 1640
15.382.1 Member Function Documentation . . . 1640
15.383 smtrat::cad::projection_compare::ProjectionComparator_impl< Args > Struct Template Reference 1641
15.383.1 Member Function Documentation . . . 1641
15.384 smtrat::cad::ProjectionGlobalInformation Class Reference . . . 1641
15.384.1 Member Typedef Documentation . . . 1641
15.384.2 Member Function Documentation . . . 1641
15.384.3 Field Documentation . . . 1642
15.385 smtrat::cad::ProjectionInformation Class Reference . . . 1642
15.385.1 Member Function Documentation . . . 1642
15.386 smtrat::cad::ProjectionLevelInformation Class Reference . . . 1644
15.386.1 Member Function Documentation . . . 1644
15.387 smtrat::cad::ProjectionOperator Struct Reference . . . 1645
15.387.1 Member Function Documentation . . . 1645
15.388 smtrat::cad::ProjectionPolynomialInformation Class Reference . . . 1645
15.388.1 Member Function Documentation . . . 1646
15.389 smtrat::cadcells::datastructures::Projections Class Reference . . . 1646
15.389.1 Detailed Description . . . 1647
15.389.2 Constructor & Destructor Documentation . . . 1647
15.389.3 Member Function Documentation . . . 1647

15.390 smtrat::cadcells::operators::PropertiesSet< Op > Struct Template Reference 1649
15.391 smtrat::cadcells::operators::PropertiesSet< op::mccallum > Struct Template Reference 1650
15.391.1 Member Typedef Documentation 1650
15.392 smtrat::cadcells::datastructures::PropertiesT< Ts > Struct Template Reference 1650
15.392.1 Detailed Description 1650
15.393 smtrat::cadcells::datastructures::PropertiesT< T, Ts... > Struct Template Reference 1650
15.393.1 Field Documentation 1650
15.394 smtrat::cadcells::datastructures::PropertiesTContent< T, is_flag > Struct Template Reference . . 1651
15.395 smtrat::cadcells::datastructures::PropertiesTContent< T, false > Struct Template Reference . . . 1651
15.395.1 Member Typedef Documentation 1651
15.396 smtrat::cadcells::datastructures::PropertiesTContent< T, true > Struct Template Reference . . . 1651
15.396.1 Member Typedef Documentation 1651
15.397 smtrat::cadcells::datastructures::property_hash< T > Struct Template Reference 1651
15.397.1 Member Function Documentation 1651
15.398 smtrat::PseudoBoolEncoder Class Reference 1651
15.398.1 Detailed Description 1652
15.398.2 Member Function Documentation 1652
15.398.3 Field Documentation 1652
15.399 smtrat::PseudoBoolNormalizer Class Reference 1653
15.399.1 Member Function Documentation 1653
15.400 smtrat::parser::QEParser Struct Reference 1653
15.400.1 Constructor & Destructor Documentation 1653
15.400.2 Member Function Documentation 1654
15.400.3 Field Documentation 1654
15.401 smtrat::parser::QualifiedIdentifierParser Struct Reference 1654
15.401.1 Constructor & Destructor Documentation 1654
15.401.2 Member Function Documentation 1654
15.401.3 Field Documentation 1654
15.402 smtrat::expression::QuantifierExpression Struct Reference 1655
15.402.1 Constructor & Destructor Documentation 1655
15.402.2 Field Documentation 1655
15.403 smtrat::parser::QuantifierParser Struct Reference 1655
15.403.1 Constructor & Destructor Documentation 1655
15.404 Minisat::Queue< T > Class Template Reference 1656
15.404.1 Member Typedef Documentation 1656
15.404.2 Constructor & Destructor Documentation 1656
15.404.3 Member Function Documentation 1656
15.405 smtrat::mcsat::icp::QueueEntry Struct Reference 1657
15.405.1 Field Documentation 1657
15.406 benchmax::RandomizationAdaptor< T > Class Template Reference 1657
15.406.1 Detailed Description 1657
15.406.2 Constructor & Destructor Documentation 1657

15.406.3 Member Function Documentation . . . 1658
15.407 smtrat::RationalCapsule Class Reference . . . 1658
15.407.1 Constructor & Destructor Documentation . . . 1658
15.407.2 Member Function Documentation . . . 1658
15.407.3 Field Documentation . . . 1658
15.408 smtrat::parser::RationalPolicies Struct Reference . . . 1659
15.408.1 Detailed Description . . . 1659
15.408.2 Member Function Documentation . . . 1659
15.409 smtrat::fixedsize_allocator< T >::rebind< U > Struct Template Reference . . . 1659
15.409.1 Member Typedef Documentation . . . 1659
15.410 smtrat::mcsat::onecellcad::recursive::RecursiveCAD Class Reference . . . 1659
15.410.1 Member Function Documentation . . . 1660
15.410.2 Field Documentation . . . 1664
15.411 smtrat::cad::projection::Reducta< Poly > Struct Template Reference . . . 1664
15.411.1 Detailed Description . . . 1664
15.411.2 Constructor & Destructor Documentation . . . 1664
15.411.3 Field Documentation . . . 1665
15.412 Minisat::RegionAllocator< T > Class Template Reference . . . 1665
15.412.1 Member Typedef Documentation . . . 1665
15.412.2 Member Enumeration Documentation . . . 1665
15.412.3 Constructor & Destructor Documentation . . . 1665
15.412.4 Member Function Documentation . . . 1666
15.413 smtrat::NNFRewriter::remove_xor_first_arg Struct Reference . . . 1667
15.413.1 Constructor & Destructor Documentation . . . 1667
15.414 smtrat::ReplaceVariablesRewriter Struct Reference . . . 1667
15.414.1 Member Typedef Documentation . . . 1667
15.414.2 Constructor & Destructor Documentation . . . 1668
15.414.3 Member Function Documentation . . . 1668
15.415 delta::ErrorHandler::result< typename > Struct Template Reference . . . 1668
15.415.1 Member Typedef Documentation . . . 1668
15.416 smtrat::parser::ErrorHandler::result< typename > Struct Template Reference . . . 1668
15.416.1 Member Typedef Documentation . . . 1668
15.417 smtrat::cad::preprocessor::ResultantRule Class Reference . . . 1669
15.417.1 Constructor & Destructor Documentation . . . 1669
15.417.2 Member Function Documentation . . . 1669
15.418 benchmax::Results Class Reference . . . 1669
15.418.1 Detailed Description . . . 1669
15.418.2 Member Function Documentation . . . 1669
15.419 smtrat::RNSEncoder Class Reference . . . 1670
15.419.1 Constructor & Destructor Documentation . . . 1670
15.419.2 Member Function Documentation . . . 1670
15.419.3 Field Documentation . . . 1671

15.420 smtrat::mcsat::onecellcad::firstlevel::root_indexer Class Reference . . . 1671
15.420.1 Member Function Documentation . . . 1672
15.420.2 Friends And Related Function Documentation . . . 1672
15.421 smtrat::cadcells::operators::properties::root_well_def Struct Reference . . . 1672
15.421.1 Member Function Documentation . . . 1672
15.421.2 Field Documentation . . . 1673
15.422 smtrat::mcsat::arithmetic::RootIndexer Class Reference . . . 1673
15.422.1 Member Typedef Documentation . . . 1673
15.422.2 Member Function Documentation . . . 1673
15.423 smtrat::cad::Sample Class Reference . . . 1674
15.423.1 Constructor & Destructor Documentation . . . 1674
15.423.2 Member Function Documentation . . . 1675
15.423.3 Friends And Related Function Documentation . . . 1676
15.424 smtrat::cad::sample_compare::SampleComparator< Iterator, Strategy > Struct Template Reference . . . 1676
15.425 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LS > Struct Template Reference . . . 1676
15.425.1 Member Function Documentation . . . 1676
15.426 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LT > Struct Template Reference . . . 1676
15.426.1 Member Function Documentation . . . 1677
15.427 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LTA > Struct Template Reference . . . 1677
15.427.1 Member Function Documentation . . . 1677
15.428 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LTS > Struct Template Reference . . . 1677
15.428.1 Member Function Documentation . . . 1677
15.429 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LTSA > Struct Template Reference . . . 1677
15.429.1 Member Function Documentation . . . 1677
15.430 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::S > Struct Template Reference . . . 1677
15.430.1 Member Function Documentation . . . 1678
15.431 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::T > Struct Template Reference . . . 1678
15.431.1 Member Function Documentation . . . 1678
15.432 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::TLSA > Struct Template Reference . . . 1678
15.432.1 Member Function Documentation . . . 1678
15.433 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::TS > Struct Template Reference . . . 1678
15.433.1 Member Function Documentation . . . 1678
15.434 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::TSA > Struct Template Reference . . . 1679
15.434.1 Member Function Documentation . . . 1679

15.435 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Type > Struct Template Reference . . . 1679
15.435.1 Member Function Documentation . . . 1679
15.436 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Value > Struct Template Reference . . . 1679
15.436.1 Member Function Documentation . . . 1679
15.437 smtrat::cad::sample_compare::SampleComparator_impl< It, Args > Struct Template Reference . 1680
15.437.1 Member Function Documentation . . . 1680
15.438 smtrat::cadcells::datastructures::SampledDerivation< Properties > Class Template Reference . 1680
15.438.1 Detailed Description . . . 1680
15.438.2 Constructor & Destructor Documentation . . . 1681
15.438.3 Member Function Documentation . . . 1681
15.439 smtrat::SampledDerivationRefCompare Struct Reference . . . 1683
15.439.1 Member Function Documentation . . . 1683
15.440 smtrat::cad::SampleIteratorQueue< Iterator, Comparator > Class Template Reference . . . 1683
15.440.1 Member Function Documentation . . . 1683
15.441 smtrat::sampling< S > Struct Template Reference . . . 1684
15.441.1 Member Function Documentation . . . 1685
15.442 smtrat::sampling< SamplingAlgorithm::LOWER_UPPER_BETWEEN_SAMPLING > Struct Template Reference . . . 1685
15.442.1 Member Function Documentation . . . 1685
15.443 smtrat::SATModule< Settings > Class Template Reference . . . 1685
15.443.1 Detailed Description . . . 1691
15.443.2 Member Typedef Documentation . . . 1692
15.443.3 Member Enumeration Documentation . . . 1692
15.443.4 Constructor & Destructor Documentation . . . 1692
15.443.5 Member Function Documentation . . . 1692
15.443.6 Friends And Related Function Documentation . . . 1718
15.443.7 Field Documentation . . . 1719
15.444 smtrat::parser::ScriptParser< Callee > Struct Template Reference . . . 1721
15.444.1 Constructor & Destructor Documentation . . . 1722
15.444.2 Member Function Documentation . . . 1722
15.444.3 Field Documentation . . . 1722
15.445 smtrat::parser::ParserState::ScriptScope Struct Reference . . . 1724
15.445.1 Constructor & Destructor Documentation . . . 1724
15.445.2 Member Function Documentation . . . 1724
15.446 smtrat::mcsat::onecellcad::firstlevel::section Struct Reference . . . 1724
15.446.1 Field Documentation . . . 1724
15.447 smtrat::onecellcad::recursive::Section Struct Reference . . . 1725
15.447.1 Detailed Description . . . 1725
15.447.2 Field Documentation . . . 1725
15.448 smtrat::mcsat::onecellcad::Section Struct Reference . . . 1725
15.448.1 Detailed Description . . . 1725

15.448.2 Field Documentation . . . 1726
15.449 smtrat::mcsat::onecellcad::levelwise::SectionHeuristic1 Struct Reference . . . 1726
15.449.1 Field Documentation . . . 1726
15.450 smtrat::mcsat::onecellcad::levelwise::SectionHeuristic2 Struct Reference . . . 1726
15.450.1 Field Documentation . . . 1726
15.451 smtrat::mcsat::onecellcad::levelwise::SectionHeuristic3 Struct Reference . . . 1726
15.451.1 Field Documentation . . . 1726
15.452 smtrat::mcsat::onecellcad::firstlevel::sector Struct Reference . . . 1726
15.452.1 Field Documentation . . . 1727
15.453 smtrat::mcsat::onecellcad::Sector Struct Reference . . . 1727
15.453.1 Detailed Description . . . 1727
15.453.2 Field Documentation . . . 1728
15.454 smtrat::onecellcad::recursive::Sector Struct Reference . . . 1728
15.454.1 Detailed Description . . . 1728
15.454.2 Member Function Documentation . . . 1728
15.454.3 Field Documentation . . . 1728
15.455 smtrat::mcsat::onecellcad::levelwise::SectorHeuristic1 Struct Reference . . . 1728
15.455.1 Field Documentation . . . 1729
15.456 smtrat::mcsat::onecellcad::levelwise::SectorHeuristic2 Struct Reference . . . 1729
15.456.1 Field Documentation . . . 1729
15.457 smtrat::mcsat::onecellcad::levelwise::SectorHeuristic3 Struct Reference . . . 1729
15.457.1 Field Documentation . . . 1729
15.458 smtrat::mcsat::SequentialAssignment< Backends > Struct Template Reference . . . 1729
15.459 smtrat::mcsat::SequentialExplanation< Backends > Struct Template Reference . . . 1729
15.460 delta::Settings Class Reference . . . 1729
15.460.1 Detailed Description . . . 1729
15.460.2 Constructor & Destructor Documentation . . . 1729
15.460.3 Member Function Documentation . . . 1730
15.460.4 Friends And Related Function Documentation . . . 1731
15.461 benchmax::settings::Settings Struct Reference . . . 1731
15.461.1 Detailed Description . . . 1731
15.462 smtrat::settings::Settings Struct Reference . . . 1732
15.463 smtrat::SettingsComponents Class Reference . . . 1732
15.463.1 Member Function Documentation . . . 1732
15.464 benchmax::SettingsParser Class Reference . . . 1732
15.464.1 Detailed Description . . . 1732
15.465 smtrat::SettingsParser Class Reference . . . 1732
15.466 smtrat::CNFerModule::SettingsType Struct Reference . . . 1732
15.466.1 Field Documentation . . . 1732
15.467 smtrat::parser::SExpressionParser Struct Reference . . . 1732
15.467.1 Constructor & Destructor Documentation . . . 1733
15.467.2 Field Documentation . . . 1733

15.468 smtrat::parser::SExpressionSequence< T > Struct Template Reference . . . 1733
15.468.1 Constructor & Destructor Documentation . . . 1733
15.468.2 Field Documentation . . . 1733
15.469 smtrat::ShortFormulaEncoder Class Reference . . . 1734
15.469.1 Constructor & Destructor Documentation . . . 1734
15.469.2 Member Function Documentation . . . 1734
15.469.3 Field Documentation . . . 1735
15.470 benchmax::simple_parser Class Reference . . . 1735
15.470.1 Constructor & Destructor Documentation . . . 1735
15.470.2 Member Function Documentation . . . 1735
15.471 smtrat::parser::Theories::SimpleSortAdder Struct Reference . . . 1736
15.471.1 Detailed Description . . . 1736
15.471.2 Constructor & Destructor Documentation . . . 1736
15.471.3 Member Function Documentation . . . 1736
15.471.4 Field Documentation . . . 1736
15.472 smtrat::parser::SimpleSymbolParser Struct Reference . . . 1736
15.472.1 Detailed Description . . . 1736
15.472.2 Constructor & Destructor Documentation . . . 1736
15.472.3 Field Documentation . . . 1737
15.473 smtrat::expression::simplifier::Simplifier Class Reference . . . 1737
15.473.1 Member Function Documentation . . . 1737
15.474 smtrat::expression::simplifier::SimplifierChainCaller< chainID > Struct Template Reference . . . 1737
15.474.1 Member Function Documentation . . . 1737
15.474.2 Field Documentation . . . 1737
15.475 smtrat::expression::simplifier::SimplifierChainCaller< 0 > Struct Template Reference . . . 1737
15.475.1 Member Function Documentation . . . 1737
15.476 smtrat::expression::simplifier::SingletonSimplifier Struct Reference . . . 1738
15.476.1 Member Function Documentation . . . 1738
15.477 smtrat::cad::sample_compare::size Struct Reference . . . 1739
15.478 smtrat::parser::Skipper Struct Reference . . . 1739
15.478.1 Constructor & Destructor Documentation . . . 1740
15.478.2 Field Documentation . . . 1740
15.479 delta::Skipper Struct Reference . . . 1740
15.479.1 Detailed Description . . . 1740
15.479.2 Constructor & Destructor Documentation . . . 1740
15.479.3 Field Documentation . . . 1740
15.480 benchmax::SlurmBackend Class Reference . . . 1740
15.480.1 Detailed Description . . . 1741
15.480.2 Member Function Documentation . . . 1741
15.480.3 Field Documentation . . . 1742
15.481 benchmax::settings::SlurmBackendSettings Struct Reference . . . 1743
15.481.1 Detailed Description . . . 1743

15.481.2 Field Documentation . . . 1743
15.482 smtrat::parser::SMTLIBParser Class Reference . . . 1744
15.482.1 Constructor & Destructor Documentation . . . 1744
15.482.2 Member Function Documentation . . . 1745
15.482.3 Field Documentation . . . 1747
15.483 benchmax::SMTRAT Class Reference . . . 1747
15.483.1 Detailed Description . . . 1748
15.483.2 Constructor & Destructor Documentation . . . 1748
15.483.3 Member Function Documentation . . . 1748
15.483.4 Field Documentation . . . 1750
15.484 benchmax::SMTRAT_Analyzer Class Reference . . . 1750
15.484.1 Detailed Description . . . 1751
15.484.2 Constructor & Destructor Documentation . . . 1751
15.484.3 Member Function Documentation . . . 1751
15.484.4 Field Documentation . . . 1753
15.485 benchmax::SMTRAT_OPB Class Reference . . . 1753
15.485.1 Detailed Description . . . 1754
15.485.2 Constructor & Destructor Documentation . . . 1754
15.485.3 Member Function Documentation . . . 1754
15.485.4 Field Documentation . . . 1755
15.486 smtrat::execution::SoftAssertion Struct Reference . . . 1756
15.486.1 Field Documentation . . . 1756
15.487 smtrat::settings::SolverSettings Struct Reference . . . 1756
15.487.1 Field Documentation . . . 1756
15.488 smtrat::parser::SortedVariableParser Struct Reference . . . 1757
15.488.1 Constructor & Destructor Documentation . . . 1757
15.488.2 Field Documentation . . . 1757
15.489 smtrat::parser::SortParser Struct Reference . . . 1757
15.489.1 Constructor & Destructor Documentation . . . 1757
15.489.2 Member Function Documentation . . . 1757
15.490 smtrat::parser::SpecConstantParser Struct Reference . . . 1758
15.490.1 Constructor & Destructor Documentation . . . 1758
15.490.2 Field Documentation . . . 1758
15.491 smtrat::SplitSOSModule< Settings > Class Template Reference . . . 1759
15.491.1 Member Typedef Documentation . . . 1764
15.491.2 Member Enumeration Documentation . . . 1764
15.491.3 Constructor & Destructor Documentation . . . 1764
15.491.4 Member Function Documentation . . . 1764
15.491.5 Field Documentation . . . 1786
15.492 benchmax::SSHBackend Class Reference . . . 1789
15.492.1 Constructor & Destructor Documentation . . . 1789
15.492.2 Member Function Documentation . . . 1789

15.492.3 Field Documentation . . . 1791
15.493 benchmax::settings::SSHBackendSettings Struct Reference . . . 1791
15.493.1 Detailed Description . . . 1791
15.493.2 Field Documentation . . . 1791
15.494 benchmax::ssh::SSHConnection Class Reference . . . 1792
15.494.1 Detailed Description . . . 1793
15.494.2 Constructor & Destructor Documentation . . . 1793
15.494.3 Member Function Documentation . . . 1793
15.495 smtrat::vs::State Class Reference . . . 1794
15.495.1 Member Typedef Documentation . . . 1798
15.495.2 Member Enumeration Documentation . . . 1798
15.495.3 Constructor & Destructor Documentation . . . 1798
15.495.4 Member Function Documentation . . . 1799
15.496 smtrat::StaticUnionFind Struct Reference . . . 1820
15.496.1 Member Typedef Documentation . . . 1820
15.496.2 Member Function Documentation . . . 1820
15.497 smtrat::statistics::StatisticsSettings Struct Reference . . . 1821
15.497.1 Field Documentation . . . 1821
15.498 smtrat::StrategyGraph Class Reference . . . 1821
15.498.1 Constructor & Destructor Documentation . . . 1821
15.498.2 Member Function Documentation . . . 1821
15.498.3 Friends And Related Function Documentation . . . 1822
15.499 delta::String Class Reference . . . 1822
15.499.1 Detailed Description . . . 1822
15.499.2 Member Function Documentation . . . 1822
15.500 Minisat::StringOption Class Reference . . . 1823
15.500.1 Constructor & Destructor Documentation . . . 1823
15.500.2 Member Function Documentation . . . 1823
15.500.3 Field Documentation . . . 1824
15.501 smtrat::parser::StringParser Struct Reference . . . 1824
15.501.1 Detailed Description . . . 1824
15.501.2 Constructor & Destructor Documentation . . . 1824
15.501.3 Field Documentation . . . 1824
15.502 smtrat::STropModule< Settings > Class Template Reference . . . 1824
15.502.1 Member Typedef Documentation . . . 1829
15.502.2 Member Enumeration Documentation . . . 1829
15.502.3 Constructor & Destructor Documentation . . . 1830
15.502.4 Member Function Documentation . . . 1830
15.502.5 Field Documentation . . . 1851
15.503 benchmax::slurm::SubmitfileProperties Struct Reference . . . 1854
15.503.1 Detailed Description . . . 1855
15.503.2 Field Documentation . . . 1855

15.504 smtrat::vs::Substitution Class Reference . . . 1856
15.504.1 Member Enumeration Documentation . . . 1856
15.504.2 Constructor & Destructor Documentation . . . 1857
15.504.3 Member Function Documentation . . . 1858
15.505 smtrat::vs::substitutionPointerEqual Struct Reference . . . 1860
15.505.1 Member Function Documentation . . . 1860
15.506 smtrat::vs::substitutionPointerHash Struct Reference . . . 1860
15.506.1 Member Function Documentation . . . 1860
15.507 delta::SymbolParser Struct Reference . . . 1860
15.507.1 Detailed Description . . . 1860
15.507.2 Constructor & Destructor Documentation . . . 1861
15.507.3 Field Documentation . . . 1861
15.508 smtrat::parser::SymbolParser Struct Reference . . . 1861
15.508.1 Constructor & Destructor Documentation . . . 1861
15.508.2 Field Documentation . . . 1861
15.509 smtrat::SymmetryModule< Settings > Class Template Reference . . . 1861
15.509.1 Member Typedef Documentation . . . 1866
15.509.2 Member Enumeration Documentation . . . 1866
15.509.3 Constructor & Destructor Documentation . . . 1867
15.509.4 Member Function Documentation . . . 1867
15.509.5 Field Documentation . . . 1889
15.510 smtrat::lra::Tableau< Settings, T1, T2 > Class Template Reference . . . 1892
15.510.1 Constructor & Destructor Documentation . . . 1894
15.510.2 Member Function Documentation . . . 1894
15.511 smtrat::lra::TableauEntry< T1, T2 > Class Template Reference . . . 1910
15.511.1 Constructor & Destructor Documentation . . . 1911
15.511.2 Member Function Documentation . . . 1912
15.512 smtrat::mcsat::onecellcad::TagPoly Struct Reference . . . 1914
15.512.1 Detailed Description . . . 1914
15.512.2 Field Documentation . . . 1914
15.513 smtrat::Task Class Reference . . . 1914
15.513.1 Constructor & Destructor Documentation . . . 1914
15.513.2 Member Function Documentation . . . 1914
15.514 delta::TempFilenameGenerator Class Reference . . . 1915
15.514.1 Detailed Description . . . 1915
15.514.2 Constructor & Destructor Documentation . . . 1915
15.514.3 Member Function Documentation . . . 1915
15.515 smtrat::parser::TermParser Struct Reference . . . 1916
15.515.1 Member Typedef Documentation . . . 1916
15.515.2 Constructor & Destructor Documentation . . . 1916
15.515.3 Field Documentation . . . 1916
15.516 smtrat::mcsat::vs::helper::TestCandidate Struct Reference . . . 1917

15.516.1 Member Function Documentation . . . 1917
15.516.2 Field Documentation . . . 1917
15.517 smtrat::parser::Theories Struct Reference . . . 1918
15.517.1 Detailed Description . . . 1918
15.517.2 Member Typedef Documentation . . . 1918
15.517.3 Constructor & Destructor Documentation . . . 1918
15.517.4 Member Function Documentation . . . 1919
15.518 smtrat::parser::TheoryError Struct Reference . . . 1921
15.518.1 Member Function Documentation . . . 1921
15.518.2 Friends And Related Function Documentation . . . 1921
15.519 smtrat::mcsat::TheoryLevel Struct Reference . . . 1921
15.519.1 Field Documentation . . . 1921
15.520 smtrat::Module::TheoryPropagation Struct Reference . . . 1922
15.520.1 Constructor & Destructor Documentation . . . 1922
15.520.2 Member Function Documentation . . . 1923
15.520.3 Field Documentation . . . 1923
15.521 smtrat::TheoryVarSchedulerStatic< vot > Class Template Reference . . . 1923
15.521.1 Detailed Description . . . 1924
15.521.2 Constructor & Destructor Documentation . . . 1924
15.521.3 Member Function Documentation . . . 1924
15.521.4 Field Documentation . . . 1925
15.522 smtrat::ThreadPool Class Reference . . . 1927
15.522.1 Constructor & Destructor Documentation . . . 1927
15.522.2 Member Function Documentation . . . 1927
15.523 smtrat::cad::debug::TikzBasePrinter Class Reference . . . 1928
15.523.1 Member Function Documentation . . . 1928
15.523.2 Field Documentation . . . 1929
15.524 smtrat::cad::debug::TikzDAGPrinter Class Reference . . . 1929
15.524.1 Member Function Documentation . . . 1929
15.524.2 Field Documentation . . . 1930
15.525 smtrat::cad::debug::TikzHistoryPrinter Class Reference . . . 1930
15.525.1 Member Function Documentation . . . 1931
15.526 smtrat::cad::debug::TikzTreePrinter Class Reference . . . 1931
15.526.1 Member Function Documentation . . . 1932
15.526.2 Field Documentation . . . 1933
15.527 benchmax::Tool Class Reference . . . 1933
15.527.1 Detailed Description . . . 1934
15.527.2 Constructor & Destructor Documentation . . . 1934
15.527.3 Member Function Documentation . . . 1935
15.527.4 Friends And Related Function Documentation . . . 1936
15.527.5 Field Documentation . . . 1936
15.528 benchmax::settings::ToolSettings Struct Reference . . . 1937

15.528.1 Detailed Description . . . 1937
15.528.2 Field Documentation . . . 1937
15.529 smtrat::TotalizerEncoder Class Reference . . . 1938
15.529.1 Detailed Description . . . 1939
15.529.2 Constructor & Destructor Documentation . . . 1939
15.529.3 Member Function Documentation . . . 1939
15.529.4 Field Documentation . . . 1940
15.530 smtrat::TotalizerTree Class Reference . . . 1940
15.530.1 Constructor & Destructor Documentation . . . 1940
15.530.2 Member Function Documentation . . . 1940
15.531 smtrat::cad::projection_compare::type Struct Reference . . . 1940
15.532 smtrat::cad::sample_compare::type Struct Reference . . . 1940
15.533 smtrat::uf_impl::uf_data_wrapper< T, compression > Class Template Reference . . . 1941
15.533.1 Detailed Description . . . 1941
15.533.2 Member Typedef Documentation . . . 1941
15.533.3 Constructor & Destructor Documentation . . . 1941
15.533.4 Member Function Documentation . . . 1941
15.534 smtrat::uf_impl::uf_data_wrapper< void, compression > Class Template Reference . . . 1942
15.534.1 Detailed Description . . . 1942
15.534.2 Constructor & Destructor Documentation . . . 1942
15.534.3 Member Function Documentation . . . 1942
15.535 smtrat::uf_impl::uf_rank_and_class< compression > Class Template Reference . . . 1943
15.535.1 Detailed Description . . . 1943
15.535.2 Constructor & Destructor Documentation . . . 1943
15.535.3 Member Function Documentation . . . 1943
15.536 smtrat::uf_impl::uf_rank_and_class< true > Class Template Reference . . . 1944
15.536.1 Detailed Description . . . 1944
15.536.2 Constructor & Destructor Documentation . . . 1944
15.536.3 Member Function Documentation . . . 1944
15.537 smtrat::UFCegarModule< Settings > Class Template Reference . . . 1945
15.537.1 Member Typedef Documentation . . . 1950
15.537.2 Member Enumeration Documentation . . . 1950
15.537.3 Constructor & Destructor Documentation . . . 1950
15.537.4 Member Function Documentation . . . 1950
15.537.5 Field Documentation . . . 1971
15.538 smtrat::UFRewriter Struct Reference . . . 1974
15.538.1 Member Typedef Documentation . . . 1974
15.538.2 Member Function Documentation . . . 1975
15.539 smtrat::expression::UnaryExpression Struct Reference . . . 1975
15.539.1 Constructor & Destructor Documentation . . . 1975
15.539.2 Field Documentation . . . 1975
15.540 smtrat::cad::debug::UnifiedData Struct Reference . . . 1975

15.540.1 Member Function Documentation . . . 1976
15.540.2 Field Documentation . . . 1976
15.541 smtrat::parser::types::UninterpretedTheory Struct Reference . . . 1976
15.541.1 Detailed Description . . . 1976
15.541.2 Member Typedef Documentation . . . 1976
15.542 smtrat::parser::UninterpretedTheory Struct Reference . . . 1977
15.542.1 Detailed Description . . . 1977
15.542.2 Constructor & Destructor Documentation . . . 1977
15.542.3 Member Function Documentation . . . 1978
15.542.4 Field Documentation . . . 1980
15.543 smtrat::union_find< T, no_compression, MaxTracebackLength > Class Template Reference . . . 1980
15.543.1 Detailed Description . . . 1981
15.543.2 Constructor & Destructor Documentation . . . 1981
15.543.3 Member Function Documentation . . . 1982
15.544 smtrat::UnionFindInterface< T, Implementation > Struct Template Reference . . . 1984
15.544.1 Member Typedef Documentation . . . 1985
15.544.2 Constructor & Destructor Documentation . . . 1985
15.544.3 Member Function Documentation . . . 1985
15.545 smtrat::UnionFindModule< Settings > Class Template Reference . . . 1986
15.545.1 Member Typedef Documentation . . . 1991
15.545.2 Member Enumeration Documentation . . . 1991
15.545.3 Constructor & Destructor Documentation . . . 1991
15.545.4 Member Function Documentation . . . 1991
15.545.5 Field Documentation . . . 2013
15.546 smtrat::SATModule< Settings >::UnorderedClauseLookup::UnorderedClauseHasher Struct Reference . . . 2016
15.546.1 Member Function Documentation . . . 2016
15.547 smtrat::UnsatCore< Solver, Strategy > Class Template Reference . . . 2016
15.547.1 Constructor & Destructor Documentation . . . 2017
15.547.2 Member Function Documentation . . . 2017
15.548 smtrat::unsatcore::UnsatCoreBackend< Solver, Strategy > Class Template Reference . . . 2017
15.549 smtrat::unsatcore::UnsatCoreBackend< Solver, UnsatCoreStrategy::ModelExclusion > Class Template Reference . . . 2018
15.549.1 Detailed Description . . . 2018
15.549.2 Constructor & Destructor Documentation . . . 2018
15.549.3 Member Function Documentation . . . 2018
15.550 smtrat::vs::unsignedTripleCmp Struct Reference . . . 2018
15.550.1 Member Function Documentation . . . 2019
15.551 smtrat::parser::UserFunctionInstantiator Struct Reference . . . 2019
15.551.1 Constructor & Destructor Documentation . . . 2019
15.551.2 Member Function Documentation . . . 2019
15.551.3 Field Documentation . . . 2019
15.552 smtrat::validation::ValidationCollector Class Reference . . . 2020

15.552.1 Member Function Documentation . . . 2020
15.553 smtrat::validation::ValidationPoint Class Reference . . . 2020
15.553.1 Member Function Documentation . . . 2020
15.554 smtrat::validation::ValidationPrinter< SOF > Struct Template Reference . . . 2021
15.555 smtrat::validation::ValidationSettings Struct Reference . . . 2021
15.555.1 Member Function Documentation . . . 2022
15.555.2 Field Documentation . . . 2022
15.556 smtrat::lra::Value< T > Class Template Reference . . . 2022
15.556.1 Constructor & Destructor Documentation . . . 2023
15.556.2 Member Function Documentation . . . 2023
15.557 smtrat::lra::Variable< T1, T2 > Class Template Reference . . . 2030
15.557.1 Constructor & Destructor Documentation . . . 2031
15.557.2 Member Function Documentation . . . 2031
15.558 smtrat::vb::Variable< T > Class Template Reference . . . 2040
15.558.1 Detailed Description . . . 2041
15.558.2 Constructor & Destructor Documentation . . . 2041
15.558.3 Member Function Documentation . . . 2041
15.559 smtrat::vb::VariableBounds< T > Class Template Reference . . . 2043
15.559.1 Detailed Description . . . 2044
15.559.2 Member Typedef Documentation . . . 2044
15.559.3 Constructor & Destructor Documentation . . . 2045
15.559.4 Member Function Documentation . . . 2045
15.560 smtrat::VariableCapsule Class Reference . . . 2050
15.560.1 Constructor & Destructor Documentation . . . 2050
15.560.2 Member Function Documentation . . . 2050
15.560.3 Field Documentation . . . 2050
15.561 smtrat::mcsat::variableordering::VariableIDs Struct Reference . . . 2050
15.561.1 Member Function Documentation . . . 2051
15.561.2 Field Documentation . . . 2051
15.562 smtrat::VariableRewriteRule Class Reference . . . 2051
15.562.1 Constructor & Destructor Documentation . . . 2051
15.562.2 Field Documentation . . . 2051
15.563 smtrat::mcsat::VariableSelector Class Reference . . . 2052
15.563.1 Member Function Documentation . . . 2052
15.563.2 Friends And Related Function Documentation . . . 2053
15.564 smtrat::variant_hash_visitor Struct Reference . . . 2053
15.564.1 Member Function Documentation . . . 2053
15.565 smtrat::parser::conversion::VariantConverter< Res > Struct Template Reference . . . 2053
15.565.1 Detailed Description . . . 2053
15.565.2 Member Typedef Documentation . . . 2054
15.565.3 Member Function Documentation . . . 2054
15.565.4 Field Documentation . . . 2054

15.566 smtrat::VariantMap< Key, Value > Class Template Reference 2054
15.566.1 Detailed Description 2055
15.566.2 Member Function Documentation 2055
15.566.3 Field Documentation 2056
15.567 smtrat::parser::conversion::VariantVariantConverter< Res > Struct Template Reference 2057
15.567.1 Detailed Description 2057
15.567.2 Member Typedef Documentation 2057
15.567.3 Member Function Documentation 2057
15.568 smtrat::VarSchedulerBase Class Reference 2057
15.568.1 Detailed Description 2058
15.568.2 Constructor & Destructor Documentation 2058
15.568.3 Member Function Documentation 2058
15.568.4 Field Documentation 2060
15.569 smtrat::VarSchedulerFixedRandom Class Reference 2061
15.569.1 Constructor & Destructor Documentation 2062
15.569.2 Member Function Documentation 2062
15.569.3 Field Documentation 2063
15.570 smtrat::VarSchedulerMcsatActivityPreferTheory< vot > Class Template Reference 2064
15.570.1 Detailed Description 2065
15.570.2 Constructor & Destructor Documentation 2065
15.570.3 Member Function Documentation 2065
15.570.4 Field Documentation 2066
15.571 smtrat::VarSchedulerMcsatBase Class Reference 2068
15.571.1 Detailed Description 2069
15.571.2 Constructor & Destructor Documentation 2069
15.571.3 Member Function Documentation 2069
15.571.4 Field Documentation 2070
15.572 smtrat::VarSchedulerMcsatBooleanFirst< vot > Class Template Reference 2072
15.572.1 Detailed Description 2072
15.572.2 Constructor & Destructor Documentation 2072
15.572.3 Member Function Documentation 2072
15.572.4 Field Documentation 2074
15.573 smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler > Class Template Reference 2075
15.573.1 Detailed Description 2076
15.573.2 Constructor & Destructor Documentation 2076
15.573.3 Member Function Documentation 2076
15.573.4 Field Documentation 2077
15.574 smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities > Class Template Reference 2079
15.574.1 Detailed Description 2080
15.574.2 Constructor & Destructor Documentation 2080
15.574.3 Member Function Documentation 2080

15.574.4 Field Documentation . . . 2081
15.575 smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot > Class Template Reference . . . 2083
15.575.1 Detailed Description . . . 2084
15.575.2 Constructor & Destructor Documentation . . . 2084
15.575.3 Member Function Documentation . . . 2084
15.575.4 Field Documentation . . . 2085
15.576 smtrat::VarSchedulerMinisat Class Reference . . . 2087
15.576.1 Detailed Description . . . 2087
15.576.2 Constructor & Destructor Documentation . . . 2087
15.576.3 Member Function Documentation . . . 2088
15.576.4 Field Documentation . . . 2089
15.577 smtrat::VarSchedulerRandom Class Reference . . . 2090
15.577.1 Detailed Description . . . 2091
15.577.2 Constructor & Destructor Documentation . . . 2091
15.577.3 Member Function Documentation . . . 2091
15.577.4 Field Documentation . . . 2092
15.578 smtrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic > Class Template Reference . . . 2093
15.578.1 Detailed Description . . . 2094
15.578.2 Constructor & Destructor Documentation . . . 2094
15.578.3 Member Function Documentation . . . 2094
15.578.4 Field Documentation . . . 2095
15.579 Minisat::vec< T > Class Template Reference . . . 2097
15.579.1 Constructor & Destructor Documentation . . . 2097
15.579.2 Member Function Documentation . . . 2098
15.580 smtrat::parser::conversion::VectorVariantConverter< Res > Struct Template Reference . . . 2100
15.580.1 Detailed Description . . . 2100
15.580.2 Member Typedef Documentation . . . 2100
15.580.3 Member Function Documentation . . . 2100
15.581 smtrat::VSModule< Settings > Class Template Reference . . . 2101
15.581.1 Member Typedef Documentation . . . 2106
15.581.2 Member Enumeration Documentation . . . 2106
15.581.3 Constructor & Destructor Documentation . . . 2106
15.581.4 Member Function Documentation . . . 2106
15.581.5 Field Documentation . . . 2129
15.582 Minisat::Watcher Struct Reference . . . 2132
15.582.1 Detailed Description . . . 2132
15.582.2 Constructor & Destructor Documentation . . . 2132
15.582.3 Member Function Documentation . . . 2133
15.582.4 Friends And Related Function Documentation . . . 2133
15.582.5 Field Documentation . . . 2133
15.583 smtrat::ICPModule< Settings >::weights Struct Reference . . . 2133

15.583.1 Field Documentation . . . 2133
15.584 benchmax::XMLWriter Class Reference . . . 2134
15.584.1 Detailed Description . . . 2134
15.584.2 Constructor & Destructor Documentation . . . 2134
15.584.3 Member Function Documentation . . . 2134
15.585 benchmax::Z3 Class Reference . . . 2134
15.585.1 Detailed Description . . . 2135
15.585.2 Constructor & Destructor Documentation . . . 2135
15.585.3 Member Function Documentation . . . 2135
15.585.4 Field Documentation . . . 2136

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://smtrat.↩ github.io/`.
- 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 are new to SMT-RAT and want to have a look around, we recommend reading the `manual`. The full API documentation can be found either `online`, or in the `pdf documentation`.

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!

2 Installation

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.
- `boost` for several additional libraries (automatically built locally if necessary).
- `gmp` for calculations with large numbers (automatically built locally if necessary).
- `carl` from http://smtrat.github.io/carl/ (automatically built locally if necessary).

Optional dependencies

- `ccmake` to set cmake flags.
- `doxygen` to build the documentation.
- `ginac` to enable the usage of polynomial factorization.

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

- `g++` >= 7
- `clang` >= 5

2.2 Download

Here are archived versions of SMT-RAT for download:

- latest

We mirror our master branch to github.com. If you want to use the newest bleeding edge version, you can checkout from https://github.com/smtrat/smtrat. Although we try to keep the master branch stable, there is a chance that the current revision is broken. You can check here if the current revision compiles.

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]

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.

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
```

3 System architecture

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`.

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.

Figure 1 System architecture

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.

Figure 2 Module 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.

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$\varphi\f$)` (`remove(\f$\varphi\f$)`). 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.

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 $\Sigma = \{1, \ldots, |E|\}$.

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 $R_1, \ldots, R_n \subseteq C_{rcv}(m)$ 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.

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]

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.

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.

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.

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.

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`.

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.

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.

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*.

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.

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 positition 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`.

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 `SMTRAT_DEVOPTION_Validation` must be activated, which can be easily achieved when using, e.g., `ccmake`.

- `void checkInfSubsetForMinimality(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 XYSettingsN, 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 `SMTRAT_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.

4.5 Existing module implementation

Available modules

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 NewCovering↩ Module PBGaussModule PBPPModule PFEModule SATModule STropModule SplitSOSModule SymmetryModule UFCegarModule UnionFindModule VSModule

4.6.1 BEModule

4.6.2 BVModule This module implements an encoder for bitvector formulae to propositional logic. It is described in more detail in [**?**].

4.6.3 CNFerModule Transforms its received formula into conjunctive normal form CNF.

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.

4.6.4 CSplitModule Implements solving for nonlinear integer arithmetic using incremental linearization. This module was implemented in [**?**] based on [**?**].

4.6.5 CoCoAGBModule Uses Gröbner bases for theory solving, very much like smtrat::GBModule. However, for the underlying implementation of Gröbner bases, this module does not use carl::groebner but CoCoALib.

4.6.6 CubeLIAModule Implements cube-based tests for linear integer arithmetic based on [**?**].

4.6.7 CurryModule Implements curryfiyng preprcessing as described in [**?**].

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

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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 smtrat::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 smtrat::ICPModule to infer new bounds and call the backend incrementally with growing bounds until the computed upper bounds are met.

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

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.

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 smtrat::LRAModule might need to communicate a lemma/tautology to a preceding smtrat::SATModule, if it receives a constraint with the relation symbol $\neq$ and the strategy needs for this reason to define a smtrat::SATModule at any position before an smtrat::LRAModule.

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 smtrat::LRAModule 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 smtrat::LRAModule.

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.

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.

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.

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

4.6.23 NewCADModule This module is based on the implementation in smtrat::cad. It is described in [**?**] in some detail.

4.6.24 PBGaussModule This module tries to simplify pseudo-Boolean problems using Gauss as described in [**?**].

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

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 $q(b)$ if 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.

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:

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

$\cdot$	$>$	$\geq$	$=$	$\leq$	$<$
$>$	$>$	$\geq$	$=$	$\leq$	$<$
$\geq$	$\geq$	$\geq$	$=$	$\leq$	$\leq$
$=$	$=$	$=$	$=$	$=$	$=$
$\leq$	$\leq$	$\leq$	$=$	$\geq$	$\geq$
$<$	$<$	$\leq$	$=$	$\geq$	$>$

4.6.27 SATModule This module abstracts it's received formula, being any SMT formula of the supported logics of \smtrat, 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.

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.

4.6.28 STropModule Implements the subtropical satisfiability method as described in [**?**] based on [**?**].

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

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.

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.

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

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).

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.

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.

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<VSModule<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<VSModule<VSSettings234>>(
                            addBackend<NewCADModule<NewCADSettingsFOS>>()
                        )
                    )
                )
            ).condition( &is_nonlinear ),
            addBackend<FPPModule<FPPSettings1>>(
                addBackend<SATModule<SATSettings1>>(
                    addBackend<LRAModule<LRASettings1>>()
                )
            ).condition( &is_linear )
        });
    }
};
```

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.

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 `SMTRAT_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, lazard, brown, none)

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

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>
```

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>
```

6.1.3 Quantifier elimination 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))`.

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>
```

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>
```

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`.

6.2 Embedding in other software

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

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 and ... and c_n) -> (p <= r or p > r)` where c_1, .., 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`, `VSModule`, 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.

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

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

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.

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::UFInstance f1 = smtrat::newUFInstance( 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.

6.2.4 Normalized constraints A normalized constraint has the form

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

with $n \geq 0$, the $*$ ith 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 $a_1, \ldots, a_n, d$ is 1. If all variable are integer valued the constraint is further simplified to

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

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

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

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.

6.2.5 Linking

[Todo: example how linking works]

7 Other tools

7.1 Benchmarking

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

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.

7.2.1 SMT-RAT's own delta tool

SMT-RAT has its own tool for delta debugging, see Delta for more information.

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
```

7.3 Analyzer

The `smtrat-analyzer` library can analyze static properties 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.

7.4 Preprocessing

WIP

7.5 Benchmax

Benchmax is a tool for automated benchmarking. Though it was developed and is primary 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.

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 resulst 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`.

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.

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.

7.5.1.2 Working with the results In the SMT-RAT repository, two utilites 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.

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

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.

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.

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.

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.

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.

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`.

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
```

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 multiplle XMLs into one:

```
import benchmax.evaluation as ev
df = ev.xmls_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).

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']))
```

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

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
```

7.5.5.5 Exporting data to Latex

7.5.5.5.1 Pandas dataframe to latex table `df.to_latex(buf='file.tex')`

7.5.5.5.2 matplotlib to tikz plot see `tikzplotlib`

7.6 Delta

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.

7.6.1.1 Using delta `delta` is currently contained in the SMT-RAT reopsitory 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`.

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.

8 Developers information

- Code style
- Documentation
- Settings
- Logging
- Statistics and timing
- Testing
- Validation
- Checkpoints
- Finding and Reporting Bugs
- Tools

8.1 Code style

8.1.0.1 Code formatting `ClangFormat` allows to define code style rules and format source files automatically. A `.clang-format` file is provided with the repository. Please use this file to format all sources.

8.1.0.2 Naming conventions For all new code, the following rules apply.

- type names and template parameter: `CamelCase`
- variable and function names: `snake_case`
- compiler macros and defines: `ALL_UPPERCASE`
- enum values: `UPPERCASE`
- (private) class members: start with `m_` respectively `mp_` for pointers and `mr_` for references

8.1.0.3 Headers and namespaces

The following conventions apply for new modules added to SMT-RAT and CArL. When adding code to existing structures, be consistent with existing code.

- Every file should represent a *module*.
- Every (sub-)folder has its own namespace.
- Namespaces and modules have names in `camel_case`.

8.1.0.4 C++ features

- As of now, please stick to C++17 features.
- Use `enum class` instead of `enum`.

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/`.

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.

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 isNormal() const;
```

8.2.1 Code comments

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.

8.2.1.2 Namespaces

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

8.2.1.3 Class headers

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

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.

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);
/// @}
```

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/`.

8.3 Settings

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

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.

8.4 Logging

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.

8.5 Statistics and timing

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.

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>
#ifdef 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
```

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()`.

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")

auto start = SMTRAT_TIME_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.

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.*

8.6.0.1 Utilities For creating CArL data structure in unit tests, we refer to `carl/src/util/parser/Parser.h`.

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.

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

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 sotre its reference to a member, use

```
SMTRAT_VALIDATION_INIT(channel, name, member);
```

Hint: to put it in a static member, use

```
SMTRAT_VALIDATION_INIT_STATIC(channel, name, member);
```

To an initialized validation point stored in a member, 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(member, formula, consistent);
```

To combine the two steps above, use:

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

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.

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.

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

8.8.0.2 Specyfing checkpoints Insert

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

into the code.

8.8.0.3 Setting up a reference run Create a test case in `src/test`. Include `#include <smtrat-common/smtrat-com` 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.

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 datastructures.
- 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/smtrat/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 of git commit id).
- A minimal working example.
- A description of what you would expect to happen.
- A description of what actually happens.

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 easer: currently, we need to resolve ite expressions within the theory in the parser, leading to bad performance on some examples.

10 NewCoveringModule #and

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

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.

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/New↩ CoveringModule/NewCoveringSettings.h

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.

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, con
check that definition matches the sort

12 Hierarchical Index

12.1 Class Hierarchy

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

smtrat::AbstractModuleFactory 245

smtrat::ModuleFactory< Module > 1323

smtrat::parser::AbstractTheory 246

smtrat::parser::ArithmeticTheory 251

smtrat::parser::BitvectorTheory 320

smtrat::parser::BooleanEncodingTheory 327

smtrat::parser::CoreTheory 501

smtrat::parser::UninterpretedTheory 1977

smtrat::cad::sample_compare::absvalue 249

std::allocator< T >

smtrat::fixedsize_allocator< void > 786

smtrat::analyzer::AnalysisSettings 249

smtrat::AnnotatedBVTerm 250

benchmax::slurm::ArchiveProperties 251

smtrat::parser::types::ArithmeticTheory 254

smtrat::execution::Assertion 254

smtrat::cad::preprocessor::AssignmentCollector 255

smtrat::mcsat::arithmetic::AssignmentFinder 256

smtrat::mcsat::smtaf::AssignmentFinder< Settings > 256

smtrat::mcsat::arithmetic::AssignmentFinder_detail 256

smtrat::mcsat::smtaf::AssignmentFinder_SMT 257

smtrat::cadcells::datastructures::detail::AssignmentProperties 258

smtrat::parser::Attribute 258

smtrat::AxiomFactory 260

benchmax::Backend 261

benchmax::CondorBackend 480

benchmax::LocalBackend 1143

benchmax::SlurmBackend 1740

benchmax::SSHBackend 1789

smtrat::Backend< Settings > 263

smtrat::BackendLink 266

smtrat::BackendSynchronisation 267

smtrat::Backtrackable< UnionFind > 267

smtrat::cadcells::datastructures::BaseDerivation< Properties > 269

smtrat::cad::BaseProjection< Settings > 271

smtrat::cad::ModelBasedProjection< ProjectionSettings > 1280

smtrat::cad::Projection< Settings > 1596

smtrat::qe::cad::Projection< smtrat::qe::cad::CADSettings > 1601

smtrat::cad::ModelBasedProjection< incrementality, backtracking, Settings > 1280

smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings > 1286

smtrat::cad::Projection< incrementality, backtracking, Settings > 1596

smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings > 1608

smtrat::cad::Projection< Incrementality::FULL, BT, Settings > 1614

smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings > 1620

smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings > 1627

smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings > 1633

smtrat::qe::cad::Projection< Settings > 1601

smtrat::cad::BaseProjection< smtrat::qe::cad::CADSettings > 271

smtrat::cad::BaseSettings 277

smtrat::qe::cad::CADSettings 393

smtrat::cad::Origin::BaseType 280

benchmax::BenchmarkResult 311

benchmax::BenchmarkSet 313

benchmax::settings::BenchmarkSettings 314

smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName > 315

smtrat::Bimap< Expansion, const carl::Variable, &Expansion::mRationalization, const carl::Variable, &Expansion::mDiscretization > 315

smtrat::Bimap< Linearization, const Poly, &Linearization::mNormalization, const Poly, &Linearization::mLinearization > 315

smtrat::expression::BinaryExpression 317

smtrat::datastructures::BinaryHeap< T > 318

smtrat::parser::types::BitvectorTheory 323

smtrat::BlastedConstr 323

smtrat::BlastedPoly 324

smtrat::mcsat::Bookkeeping 325

smtrat::BoolUEQRewriter 331

smtrat::lra::Bound< T1, T2 > 331

smtrat::vb::Bound< T > 342

smtrat::mcsat::fm::Bound 346

smtrat::vb::Bound< ConstraintT > 342

smtrat::vb::Bound< FormulaT > 342

smtrat::lra::Bound< LRABoundType, LRAEntryType > 331

smtrat::Branching 347

smtrat::BVAnnotation 348

smtrat::BVDirectEncoder 350

smtrat::by_address_hasher< IterType > 379

smtrat::cad::CAD< Settings > 380

smtrat::qe::cad::CAD< Settings > 383

smtrat::qe::cad::CAD< smtrat::qe::cad::CADSettings > 383

smtrat::cad::CADConstraints< BT > 384

smtrat::cad::CADConstraints< ProjectionSettings::backtracking > 384

smtrat::cad::CADConstraints< Settings::backtracking > 384

smtrat::cad::CADConstraints< smtrat::qe::cad::CADSettings ::backtracking > 384

smtrat::cad::CADCore< CH > 388

smtrat::cad::CADCore< CoreHeuristic::BySample > 388

smtrat::cad::CADCore< CoreHeuristic::EnumerateAll > 389

smtrat::cad::CADCore< CoreHeuristic::Interleave > 389

smtrat::cad::CADCore< CoreHeuristic::PreferProjection > 390

smtrat::cad::CADCore< CoreHeuristic::PreferSampling > 390

smtrat::qe::cad::CADElimination 390

smtrat::cad::CADPreprocessor 391

smtrat::cad::CADPreprocessorSettings 392

smtrat::cadcells::representation::cell< H > 395

smtrat::cadcells::representation::cell< CellHeuristic::BIGGEST_CELL > 395

smtrat::cadcells::representation::cell< CellHeuristic::CHAIN_EQ > 396

smtrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS > 396

smtrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS_EQ > 396

smtrat::cadcells::operators::properties::cell_connected 397

smtrat::cadcells::datastructures::CellDescription 397

smtrat::cadcells::datastructures::CellRepresentation< P > 399

delta::Checker 400

benchmax::slurm::ChunkedSubmitfileProperties 401

Minisat::Clause 403

smtrat::mcsat::ClauseChain 408

smtrat::sat::detail::ClauseChecker 411

smtrat::CMakeOptionPrinter 411

Minisat::CMap< T > 411

smtrat::CollectionWithOrigins< Element, Origin > 475

smtrat::CollectionWithOrigins< carl::Variable, FormulaT > 475

smtrat::ICPModule< Settings >::comp 477

smtrat::vs::Condition 477

smtrat::mcsat::fm::ConflictGenerator< Comparator > 482

smtrat::cad::ConflictGraph 483

smtrat::parser::Theories::ConstantAdder 485

smtrat::cad::CADConstraints< BT >::ConstraintComparator 485

smtrat::cad::preprocessor::ConstraintUpdate 485

smtrat::ConstrTree 486

delta::Consumer 486

smtrat::LRAModule< Settings >::Context 488

smtrat::icp::ContractionCandidate 489

smtrat::icp::contractionCandidateComp 494

smtrat::icp::ContractionCandidateManager 494

smtrat::parser::conversion::Converter< To > 496

smtrat::parser::conversion::Converter< FormulaT > 497

smtrat::parser::conversion::Converter< Poly > 497

smtrat::parser::conversion::Converter< Res > 496

smtrat::parser::conversion::Converter< types::BVTerm > 498

benchmax::settings::CoreSettings 499

smtrat::settings::CoreSettings 500

smtrat::parser::types::CoreTheory 503

smtrat::cadcells::representation::covering< H > 504

smtrat::mcsat::arithmetic::Covering 504

smtrat::cadcells::representation::covering< CoveringHeuristic::CHAIN_COVERING > 505

smtrat::cadcells::representation::covering< CoveringHeuristic::DEFAULT_COVERING > 505

smtrat::mcsat::onecellcad::firstlevel::covering_at_level 506

smtrat::cadcells::datastructures::CoveringDescription 506

smtrat::cadcells::datastructures::CoveringRepresentation< P > 507

smtrat::mcsat::onecellcad::recursive::CoverNullification 508

smtrat::CycleEnumerator< FHG, Collector > 598

benchmax::Database 598

benchmax::DBAL 600

Minisat::DeepEqual< K > 604

Minisat::DeepHash< K > 604

smtrat::mcsat::fm::DefaultComparator 605

smtrat::mcsat::vs::DefaultSettings 605

smtrat::mcsat::smtaf::DefaultSettings 605

smtrat::mcsat::fm::DefaultSettings 606

smtrat::cad::projection_compare::degree 606

smtrat::mcsat::onecellcad::recursive::DegreeAscending 606

smtrat::analyzer::DegreeCollector 606

smtrat::mcsat::onecellcad::recursive::DegreeDescending 607

smtrat::cadcells::datastructures::DelineatedDerivation< Properties > 607

smtrat::cadcells::datastructures::Delineation 609

smtrat::cadcells::representation::delineation< H > 611

smtrat::cadcells::representation::delineation< DelineationHeuristic::CHAIN > 611

smtrat::cadcells::datastructures::DelineationInterval 611

smtrat::cadcells::datastructures::DelineationRepresentation< P > 612

smtrat::mcsat::icp::Dependencies 613

smtrat::cadcells::datastructures::DerivationRef< Properties > 614

smtrat::mcsat::onecellcad::recursive::DontCoverNullification 616

smtrat::cad::debug::DotSubgraph 617

Minisat::DoubleRange 619

smtrat::DynamicPriorityQueue< T, Compare > 621

smtrat::dynarray< T > 623

smtrat::dynarray< implicit_edge_info * > 623

smtrat::dynarray< ineq_edge_info * > 623

smtrat::dynarray_allocator< T > 627

smtrat::datastructures::DynHeap< KeyType, Compare, WeightType > 627

smtrat::cad::ProjectionGlobalInformation::ECData 628

smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Edge 629

smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType > 631

smtrat::EQModule< Settings >::graph_info::edge_list_type< edge_info > 631

smtrat::EQModule< Settings >::graph_info::edge_list_type< implicit_edge_info > 631

smtrat::EQModule< Settings >::graph_info::edge_list_type< ineq_edge_info > 631

smtrat::ElementWithOrigins< Element, Origin > 633

smtrat::EQGraph< VertexName > 665

smtrat::EQGraph< carl::UVariable > 665

Minisat::Equal< K > 726

Minisat::Equal< CRef > 726

delta::ErrorHandler 726

smtrat::parser::ErrorHandler 727

smtrat::uf_impl::estimator< IteratorType, IsRandomAccess > 758

smtrat::uf_impl::estimator< IteratorType, false > 758

smtrat::execution::ExecutionState 760

smtrat::mcsat::onecellcad::recursive::Explanation< Setting1, Setting2 > 770

smtrat::mcsat::vs::Explanation 771

smtrat::mcsat::fm::Explanation< Settings > 771

smtrat::mcsat::icp::Explanation 771

smtrat::mcsat::nlsat::Explanation 772

smtrat::mcsat::onecell::Explanation 772

smtrat::mcsat::onecellcad::levelwise::Explanation< Setting1, Setting2 > 773

smtrat::mcsat::vs::ExplanationGenerator< Settings > 773

smtrat::mcsat::nlsat::ExplanationGenerator 773

smtrat::expression::Expression 774

smtrat::expression::ExpressionContent 777

smtrat::parser::ParserState::ExpressionScope 782

false_type

smtrat::is_variant< T > 1123

smtrat::mcsat::FastParallelExplanation< Backends > 784

smtrat::mcsat::variableordering::detail::FeatureCollector< Objects > 784

smtrat::impl::fixedsize_allocator_impl< T, SizeShift, LargeEnough, SmallEnough > 787

smtrat::impl::fixedsize_allocator_impl< T, roundup_log2(Size),(Size >=sizeof(void *)),(Size<=2048)>787

smtrat::impl::fixedsize_allocator_size_helper< T,(sizeof(void *) > sizeof(T) ? sizeof(void *) ↩ :sizeof(T))> 789

smtrat::fixedsize_allocator< T > 785

smtrat::impl::fixedsize_allocator_size_helper< T, Size > 789

smtrat::impl::fixedsize_allocator_impl< T, SizeShift, true, true > 787

smtrat::fixedsize_freelist< ChunkSizeShift > 789

smtrat::impl::fixedsize_freelists< SizeCurrent, SizeMax, LowEnough > 790

smtrat::impl::fixedsize_freelists< LOWER_SIZE_BOUND, UPPER_SIZE_BOUND > 790

smtrat::impl::fixedsize_allocator_freelists 787

smtrat::impl::fixedsize_freelists< SizeCurrent *2, SizeMax > 790

smtrat::impl::fixedsize_freelists< SizeCurrent, SizeMax, true > 791

smtrat::parser::FixedWidthConstant< T > 791

formula_visitor

smtrat::CollectBoolsInUEQs 474

smtrat::ICPModule< Settings >::formulaPtrComp 798

smtrat::FormulaWithOrigins 798

smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices > 801

smtrat::qe::fm::FourierMotzkinQE 834

smtrat::freelist< T > 865

smtrat::freelist< bucket > 865

smtrat::freelist< edge_info > 865

smtrat::freelist< EdgeType > 865

smtrat::freelist< entry > 865

smtrat::freelist< implicit_edge_info > 865

smtrat::freelist< ineq_edge_info > 865

smtrat::mcsat::FullParallelExplanation< Backends > 867

smtrat::cad::FullSampleComparator< Iterator, Strategy > 867

smtrat::parser::FunctionInstantiator 867

smtrat::parser::UserFunctionInstantiator 2019

smtrat::GBModuleState< Settings > 906

grammar

delta::Skipper 1740

delta::SymbolParser 1860

smtrat::parser::ScriptParser< smtrat::parser::SMTLIBParser > 1721

smtrat::parser::AttributeParser 259

smtrat::parser::AttributeValueParser 260

smtrat::parser::BinaryParser 319

smtrat::parser::HexadecimalParser 943

smtrat::parser::IdentifierParser 1014

smtrat::parser::KeywordParser 1127

smtrat::parser::QEParser 1653

smtrat::parser::QualifiedIdentifierParser 1654

smtrat::parser::ScriptParser< Callee > 1721

smtrat::parser::SExpressionParser 1732

smtrat::parser::SimpleSymbolParser 1736

smtrat::parser::Skipper 1739

smtrat::parser::SortedVariableParser 1757

smtrat::parser::SortParser 1757

smtrat::parser::SpecConstantParser 1758

smtrat::parser::StringParser 1824

smtrat::parser::SymbolParser 1861

smtrat::parser::TermParser 1916

smtrat::EQModule< Settings >::not_asserted_equality::hash 938

Minisat::Hash< K > 938

std::hash< const smtrat::expression::ExpressionContent * > 938

std::hash< smtrat::expression::BinaryExpression > 938

std::hash< smtrat::expression::Expression > 939

std::hash< smtrat::expression::ITEExpression > 939

std::hash< smtrat::expression::NaryExpression > 939

std::hash< smtrat::expression::QuantifierExpression > 939

std::hash< smtrat::expression::UnaryExpression > 939

std::hash< smtrat::lra::Variable< T1, T2 > > 940

std::hash< smtrat::vs::Substitution > 940

std::hash< std::vector< T > > 940

std::hash_combiner 941

Minisat::Heap< Comp > 941

Minisat::Heap< VarOrderLt > 941

smtrat::icp::HistoryNode 944

smtrat::icp::IcpVariable 1009

smtrat::icp::icpVariableComp 1012

smtrat::parser::Identifier 1012

smtrat::cad::debug::IDSanitizer 1014

smtrat::mcsat::fm::IgnoreCoreSettings 1014

Implementation

smtrat::UnionFindInterface< T, Implementation > 1984

smtrat::mcsat::onecellcad::firstlevel::indexed_root 1044

smtrat::parser::IndexedFunctionInstantiator 1044

smtrat::cadcells::datastructures::IndexedRoot 1045

smtrat::cadcells::datastructures::IndexedRootOrdering 1046

smtrat::InequalitiesTable< Settings > 1047

smtrat::lra::Bound< T1, T2 >::Info 1050

smtrat::mcsat::InformationGetter 1052

smtrat::parser::InstructionHandler 1055

smtrat::Executor< Strategy > 763

smtrat::parseformula::FormulaCollector 792

Minisat::Int64Range 1061

smtrat::mcsat::icp::IntervalPropagation 1120

Minisat::IntRange 1122

smtrat::is_sample_outside< S > 1123

smtrat::is_sample_outside< IsSampleOutsideAlgorithm::DEFAULT > 1123

smtrat::expression::ITEExpression 1123

iterator

benchmax::RandomizationAdaptor< T >::iterator 1124

smtrat::ModuleInput::IteratorCompare 1125

benchmax::Jobs 1125

smtrat::JunctorMerger 1126

Minisat::lbool 1127

smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound 1128

smtrat::Module::Lemma 1130

Minisat::LessThan_default< T > 1131

smtrat::cad::sample_compare::level 1131

smtrat::cad::projection_compare::level 1131

smtrat::cad::ProjectionLevelInformation::LevelInfo 1131

smtrat::LevelWiseInformation< Settings > 1135

smtrat::cad::LiftingTree< Settings > 1138

smtrat::cad::LiftingTree< smtrat::qe::cad::CADSettings > 1138

smtrat::ICPModule< Settings >::linearVariable 1142

smtrat::mcsat::ClauseChain::Link 1142

std::list< T >

smtrat::ModuleInput 1324

Minisat::Lit 1143

smtrat::Manager 1213

std::map< K, T >

smtrat::VariantMap< Key, Value > 2054

smtrat::VariantMap< std::string, Value > 2054

Minisat::Map< K, D, H, E > 1223

Minisat::Map< CRef, T, CRefHash > 1223

smtrat::mcsat::fm::MaxSizeComparator 1228

smtrat::MaxSMT< Solver, Strategy > 1228

smtrat::MaxSMT< Strategy, MaxSMTStrategy::LINEAR_SEARCH > 1228

smtrat::maxsmt::MaxSMTBackend< Solver, Strategy > 1229

smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::FU_MALIK_INCREMENTAL > 1229

smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::LINEAR_SEARCH > 1230

smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::MSU3 > 1230

smtrat::mcsat::MCSATBackend< Settings > 1262

smtrat::mcsat::MCSATBackend< typename Settings::MCSATSettings > 1262

smtrat::mcsat::MCSATMixin< Settings > 1263

smtrat::mcsat::MCSATMixin< typename Settings::MCSATSettings > 1263

smtrat::mcsat::fm::MinSizeComparator 1277

smtrat::mcsat::fm::MinVarCountComparator 1277

smtrat::cad::MISGeneration< heuristic > 1277

smtrat::Module 1293

smtrat::EQModule< EQSettingsForPreprocessing > 667

smtrat::ICPModule< ICPSettings1 > 978

smtrat::ICPModule< ICPSettings4 > 978

smtrat::LRAModule< LRASettings1 > 1148

smtrat::LRAModule< LRASettingsICP > 1148

smtrat::BVModule< Settings > 350

smtrat::CoCoAGBModule< Settings > 444

smtrat::CSplitModule< Settings > 508

smtrat::CubeLIAModule< Settings > 538

smtrat::CurryModule< Settings > 568

smtrat::EQModule< Settings > 667

smtrat::EQPreprocessingModule< Settings > 697

smtrat::FouMoModule< Settings > 804

smtrat::GBModule< Settings > 868

smtrat::ICPModule< Settings > 978

smtrat::IncWidthModule< Settings > 1015

smtrat::IntBlastModule< Settings > 1062

smtrat::IntEqModule< Settings > 1091

smtrat::LRAModule< Settings > 1148

smtrat::NewCADModule< Settings > 1336

smtrat::NewCoveringModule< Settings > 1366

smtrat::NRAILModule< Settings > 1403

smtrat::PBGaussModule< Settings > 1458

smtrat::PBPPModule< Settings > 1490

smtrat::PModule 1550

- smtrat::BEModule< Settings > 282
- smtrat::CNFerModule 413
- smtrat::EMModule< Settings > 634
- smtrat::ESModule< Settings > 727
- smtrat::FPPModule< Settings > 835
- smtrat::GBPPModule< Settings > 907
- smtrat::ICEModule< Settings > 948
- smtrat::LVEModule< Settings > 1183
- smtrat::MCBModule< Settings > 1230
- smtrat::PFEModule< Settings > 1520
- smtrat::SplitSOSModule< Settings > 1759
- smtrat::SymmetryModule< Settings > 1861

smtrat::SATModule< Settings > 1685

smtrat::STropModule< Settings > 1824

smtrat::UFCegarModule< Settings > 1945

smtrat::UnionFindModule< Settings > 1986

smtrat::VSModule< Settings > 2101

smtrat::settings::ModuleSettings 1330

smtrat::ModuleWrapper< M > 1332

smtrat::ModuleWrapper< smtrat::EQModule< EQSettingsForPreprocessing > > 1332

smtrat::expression::NaryExpression 1333

smtrat::NNFPreparation 1395

smtrat::NNFRewriter 1396

benchmax::ssh::Node 1397

delta::Node 1398

delta::NodePrinter< pretty > 1402

smtrat::mcsat::onecellcad::recursive::NoHeuristic 1403

smtrat::lra::Numeric 1433

smtrat::execution::Objective 1440

Minisat::OccLists< Idx, Vec, Deleted > 1440

Minisat::OccLists< Minisat::Lit, Minisat::vec< Minisat::Watcher >, WatcherDeleted > 1440

smtrat::mcsat::onecellcad::OneCellCAD 1441

smtrat::mcsat::onecellcad::levelwise::LevelwiseCAD 1133

smtrat::mcsat::onecellcad::recursive::RecursiveCAD 1659

benchmax::settings::OperationSettings 1443

smtrat::Optimization< Solver > 1444

smtrat::Optimization< Strategy > 1444

Minisat::Option 1445

Minisat::BoolOption 329

Minisat::DoubleOption 617

Minisat::IntOption 1121

Minisat::StringOption 1823

Minisat::Option::OptionLt 1446

smtrat::cad::Origin 1447

smtrat::cad::preprocessor::Origins 1450

smtrat::OrPathShortener 1451

Minisat::OutOfMemoryException 1452

smtrat::parser::OutputWrapper 1452

pair

smtrat::datastructures::OrderPair< T1, T2, reversed > 1447

smtrat::datastructures::OrderPair< T1, T2, true > 1447

Minisat::Map< K, D, H, E >::Pair 1452

smtrat::pairhash< F, S, FirstHasher, SecondHasher > 1453

smtrat::mcsat::ParallelExplanation< Backends > 1453

delta::Parser 1453

smtrat::parser::ParserSettings 1454

smtrat::parser::ParserState 1455

smtrat::PersistentUnionFind 1519

smtrat::cadcells::operators::properties::poly_irreducible_sgn_inv 1581

smtrat::cadcells::operators::properties::poly_ord_inv 1581

smtrat::cadcells::operators::properties::poly_pdel 1582

smtrat::cadcells::operators::properties::poly_sgn_inv 1582

smtrat::cad::ProjectionPolynomialInformation::PolyInfo 1583

smtrat::cad::PolynomialComparator< PolynomialGetter > 1583

smtrat::cad::PolynomialComparator< smtrat::cad::BaseProjection > 1583

smtrat::cad::PolynomialLiftingQueue< PolynomialGetter > 1584

smtrat::cad::PolynomialLiftingQueue< smtrat::cad::BaseProjection > 1584

smtrat::cadcells::datastructures::PolyPool 1585

smtrat::cadcells::datastructures::detail::PolyProperties 1586

smtrat::cadcells::datastructures::PolyRef 1587

smtrat::PolyTree 1587

smtrat::PolyTreeContent 1588

smtrat::cad::Preprocessor 1589

smtrat::cad::PreprocessorSettings 1591

benchmax::settings::PresetSettings 1591

std::priority_queue< T >

- **smtrat::PriorityQueue< Iterator, Comparator >** 1592
- **smtrat::PriorityQueue< Iterator, smtrat::cad::FullSampleComparator >** 1592
- **smtrat::PriorityQueue< Iterator, smtrat::cad::sample_compare::SampleComparator >** 1592
- **smtrat::PriorityQueue< QueueEntry, ProjectionCandidateComparator >** 1592
- **smtrat::PriorityQueue< QueueEntry, smtrat::cad::PolynomialComparator >** 1592
- **smtrat::PriorityQueue< T, Compare >** 1592

delta::Producer 1594

delta::ProgressBar 1594

smtrat::cad::projection_compare::ProjectionComparator< Strategy > 1639

smtrat::cad::projection_compare::ProjectionComparator< Settings::projectionComparator > 1639

smtrat::cad::projection_compare::ProjectionComparator_impl< Args > 1641

smtrat::cad::projection_compare::ProjectionComparator_impl< degree, lt > 1641

- **smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::D >** 1639

smtrat::cad::projection_compare::ProjectionComparator_impl< level, gt, degree, lt > 1641

- **smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::LD >** 1640

smtrat::cad::projection_compare::ProjectionComparator_impl< level, lt, degree, lt > 1641

- **smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::ID >** 1639

smtrat::cad::projection_compare::ProjectionComparator_impl< type, gt, degree, lt > 1641

- **smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::PD >** 1640

smtrat::cad::projection_compare::ProjectionComparator_impl< type, lt, degree, lt > 1641

- **smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::SD >** 1640

smtrat::cad::ProjectionGlobalInformation 1641

smtrat::cad::ProjectionInformation 1642

smtrat::cad::ProjectionLevelInformation 1644

smtrat::cad::ProjectionOperator 1645

smtrat::cad::ProjectionPolynomialInformation 1645

smtrat::cadcells::datastructures::Projections 1646

smtrat::cadcells::operators::PropertiesSet< Op > 1649

smtrat::cadcells::operators::PropertiesSet< op::mccallum > 1650

smtrat::cadcells::datastructures::PropertiesT< Ts > 1650

smtrat::cadcells::datastructures::PropertiesT< Ts... > 1650

smtrat::cadcells::datastructures::PropertiesT< T, Ts... > 1650

smtrat::cadcells::datastructures::PropertiesTContent< T, is_flag > 1651

smtrat::cadcells::datastructures::PropertiesTContent< T, false > 1651

smtrat::cadcells::datastructures::PropertiesTContent< T, T::is_flag > 1651

smtrat::cadcells::datastructures::PropertiesTContent< T, true > 1651

smtrat::cadcells::datastructures::property_hash< T > 1651

smtrat::PseudoBoolEncoder 1651

smtrat::CardinalityEncoder 394

smtrat::ExactlyOneCommanderEncoder 758

smtrat::LongFormulaEncoder 1146

smtrat::MixedSignEncoder 1279

smtrat::RNSEncoder 1670

smtrat::ShortFormulaEncoder 1734

smtrat::TotalizerEncoder 1938

smtrat::PseudoBoolNormalizer 1653

smtrat::expression::QuantifierExpression 1655

Minisat::Queue< T > 1656

smtrat::mcsat::icp::QueueEntry 1657

benchmax::RandomizationAdaptor< T > 1657

smtrat::RationalCapsule 1658

real_parser

smtrat::parser::DecimalParser 604

real_policies

smtrat::parser::RationalPolicies 1659

smtrat::fixedsize_allocator< T >::rebind< U > 1659

Minisat::RegionAllocator< T > 1665

Minisat::RegionAllocator< uint32_t > 1665

Minisat::ClauseAllocator 406

smtrat::NNFRewriter::remove_xor_first_arg 1667

smtrat::ReplaceVariablesRewriter 1667

delta::ErrorHandler::result< typename > 1668

smtrat::parser::ErrorHandler::result< typename > 1668

smtrat::cad::preprocessor::ResultantRule 1669

benchmax::Results 1669

smtrat::mcsat::onecellcad::firstlevel::root_indexer 1671

smtrat::cadcells::operators::properties::root_well_def 1672

smtrat::mcsat::arithmetic::RootIndexer 1673

smtrat::cad::Sample 1674

smtrat::cad::sample_compare::SampleComparator< Iterator, Strategy > 1676

smtrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::T > 867

smtrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::Type > 867

smtrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::Value > 867

smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::T > 1676

smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Type > 1676

smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Type > 1679

smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Value > 1676

smtrat::cad::sample_compare::SampleComparator_impl< It, Args > 1680

smtrat::cad::sample_compare::SampleComparator_impl< Iterator, level, gt, size, lt > 1680

smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LS > 1676

smtrat::cad::sample_compare::SampleComparator_impl< Iterator, level, gt, type, gt > 1680

smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LT > 1676

smtrat::cad::sample_compare::SampleComparator_impl< Iterator, level, gt, type, gt, absvalue, lt > 1680

smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LTA > 1677

smtrat::cad::sample_compare::SampleComparator_impl< Iterator, level, gt, type, gt, size, lt > 1680

smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LTS > 1677

smtrat::cad::sample_compare::SampleComparator_impl< Iterator, level, gt, type, gt, size, lt, absvalue, lt > 1680

smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LTSA >1677

smtrat::cad::sample_compare::SampleComparator_impl< Iterator, size, lt > 1680

smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::S > 1677

smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Value >1679

smtrat::cad::sample_compare::SampleComparator_impl< Iterator, type, gt > 1680

smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::T > 1678

smtrat::cad::sample_compare::SampleComparator_impl< Iterator, type, gt, level, gt, size, lt, absvalue, lt > 1680

smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::TLSA >1678

smtrat::cad::sample_compare::SampleComparator_impl< Iterator, type, gt, size, lt > 1680

smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::TS > 1678

smtrat::cad::sample_compare::SampleComparator_impl< Iterator, type, gt, size, lt, absvalue, lt > 1680

smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::TSA > 1679

smtrat::cadcells::datastructures::SampledDerivation< Properties > 1680

smtrat::SampledDerivationRefCompare 1683

smtrat::cad::SampleIteratorQueue< Iterator, Comparator > 1683

smtrat::cad::SampleIteratorQueue< Iterator, smtrat::cad::FullSampleComparator > 1683

smtrat::cad::SampleIteratorQueue< Iterator, smtrat::cad::sample_compare::SampleComparator > 1683

smtrat::sampling< S > 1684

smtrat::sampling< SamplingAlgorithm::LOWER_UPPER_BETWEEN_SAMPLING > 1685

smtrat::parser::ParserState::ScriptScope 1724

smtrat::mcsat::onecellcad::firstlevel::section 1724

smtrat::onecellcad::recursive::Section 1725

smtrat::mcsat::onecellcad::Section 1725

smtrat::mcsat::onecellcad::levelwise::SectionHeuristic1 1726

smtrat::mcsat::onecellcad::levelwise::SectionHeuristic2 1726

smtrat::mcsat::onecellcad::levelwise::SectionHeuristic3 1726

smtrat::mcsat::onecellcad::firstlevel::sector 1726

smtrat::mcsat::onecellcad::Sector 1727

smtrat::onecellcad::recursive::Sector 1728

smtrat::mcsat::onecellcad::levelwise::SectorHeuristic1 1728

smtrat::mcsat::onecellcad::levelwise::SectorHeuristic2 1729

smtrat::mcsat::onecellcad::levelwise::SectorHeuristic3 1729

smtrat::mcsat::SequentialAssignment< Backends > 1729

smtrat::mcsat::SequentialExplanation< Backends > 1729

delta::Settings 1729

Settings

- **benchmax::settings::Settings** 1731
- **smtrat::settings::Settings** 1732

SettingsParser

- **benchmax::SettingsParser** 1732
- **smtrat::SettingsParser** 1732

smtrat::CNFerModule::SettingsType 1732

benchmax::simple_parser 1735

smtrat::parser::Theories::SimpleSortAdder 1736

smtrat::expression::simplifier::Simplifier 1737

smtrat::expression::simplifier::SimplifierChainCaller< chainID > 1737

smtrat::expression::simplifier::SimplifierChainCaller< 0 > 1737

Singleton

- **benchmax::settings::Settings** 1731
- **benchmax::SettingsParser** 1732
- **smtrat::expression::ExpressionPool** 780
- **smtrat::LOG** 1146
- **smtrat::MonomialMappingByVariablePool** 1333
- **smtrat::PolyTreePool** 1589
- **smtrat::resource::Limiter** 1141
- **smtrat::settings::Settings** 1732
- **smtrat::SettingsComponents** 1732
- **smtrat::SettingsParser** 1732
- **smtrat::validation::ValidationCollector** 2020

smtrat::cad::sample_compare::size 1739

benchmax::settings::SlurmBackendSettings 1743

smtrat::parser::SMTLIBParser 1744

smtrat::execution::SoftAssertion 1756

smtrat::settings::SolverSettings 1756

benchmax::settings::SSHBackendSettings 1791

benchmax::ssh::SSHConnection 1792

smtrat::vs::State 1794

static_visitor

- **smtrat::expression::ExpressionConverter** 778
- **smtrat::expression::ExpressionModifier** 779
- **smtrat::expression::ExpressionTypeChecker< T >** 782
- **smtrat::expression::ExpressionVisitor** 782
- **smtrat::expression::simplifier::BaseSimplifier** 278
  - **smtrat::expression::simplifier::DuplicateSimplifier** 619
  - **smtrat::expression::simplifier::MergeSimplifier** 1270
  - **smtrat::expression::simplifier::NegationSimplifier** 1334

smtrat::expression::simplifier::SingletonSimplifier 1738

smtrat::parser::conversion::VariantConverter< Res > 2053

smtrat::parser::conversion::VariantVariantConverter< Res > 2057

smtrat::parser::Instantiator< V, T > 1053

smtrat::variant_hash_visitor 2053

smtrat::parser::conversion::VariantConverter< types::BVTerm > 2053

smtrat::parser::conversion::VariantVariantConverter< types::AttributeValue > 2057

smtrat::parser::conversion::VariantVariantConverter< types::TermType > 2057

smtrat::StaticUnionFind 1820

Statistics

smtrat::analyzer::AnalyzerStatistics 249

smtrat::Module::ModuleStatistics 1331

smtrat::statistics::StatisticsSettings 1821

smtrat::StrategyGraph 1821

delta::String 1822

benchmax::slurm::SubmitfileProperties 1854

smtrat::vs::Substitution 1856

smtrat::vs::substitutionPointerEqual 1860

smtrat::vs::substitutionPointerHash 1860

symbols

smtrat::parser::LogicParser 1146

smtrat::parser::QuantifierParser 1655

smtrat::lra::Tableau< Settings, T1, T2 > 1892

smtrat::lra::Tableau< typename LRASettings1 ::Tableau_settings, LRABoundType, LRAEntryType > 1892

smtrat::lra::Tableau< typename LRASettingsICP ::Tableau_settings, LRABoundType, LRAEntry↩ Type > 1892

smtrat::lra::Tableau< typename Settings::Tableau_settings, LRABoundType, LRAEntryType > 1892

smtrat::lra::TableauEntry< T1, T2 > 1910

smtrat::lra::TableauEntry< LRABoundType, LRAEntryType > 1910

smtrat::mcsat::onecellcad::TagPoly 1914

smtrat::Task 1914

delta::TempFilenameGenerator 1915

smtrat::mcsat::vs::helper::TestCandidate 1917

smtrat::parser::Theories 1918

smtrat::parser::TheoryError 1921

smtrat::mcsat::TheoryLevel 1921

smtrat::Module::TheoryPropagation 1922

smtrat::ThreadPool 1927

smtrat::cad::debug::TikzBasePrinter 1928

smtrat::cad::debug::TikzDAGPrinter 1929

smtrat::cad::debug::TikzTreePrinter 1931

smtrat::cad::debug::TikzHistoryPrinter 1930

benchmax::Tool 1933

benchmax::MathSAT 1225

benchmax::Minisat 1271

benchmax::Minisatp 1274

benchmax::SMTRAT 1747

benchmax::SMTRAT_OPB 1753

benchmax::SMTRAT_Analyzer 1750

benchmax::Z3 2134

benchmax::settings::ToolSettings 1937

smtrat::TotalizerTree 1940

smtrat::cad::projection_compare::type 1940

smtrat::cad::sample_compare::type 1940

smtrat::uf_impl::uf_data_wrapper< T, compression > 1941

smtrat::uf_impl::uf_data_wrapper< void, compression > 1942

smtrat::uf_impl::uf_rank_and_class< compression > 1943

smtrat::uf_impl::uf_rank_and_class< true > 1944

smtrat::UFRewriter 1974

uint_parser

smtrat::parser::NumeralParser 1433

smtrat::expression::UnaryExpression 1975

smtrat::cad::debug::UnifiedData 1975

smtrat::parser::types::UninterpretedTheory 1976

smtrat::union_find< T, no_compression, MaxTracebackLength > 1980

smtrat::union_find< g_iterator, !EQSettingsForPreprocessing ::uf_use_compression, EQSettings↩ ForPreprocessing ::uf_max_traceback_length > 1980

smtrat::union_find< g_iterator, !Settings::uf_use_compression, Settings::uf_max_traceback_length > 1980

smtrat::union_find< uf_component_entry, !EQSettingsForPreprocessing ::uf_use_compression, EQSettingsForPreprocessing ::uf_max_traceback_length > 1980

smtrat::union_find< uf_component_entry, !Settings::uf_use_compression, Settings::uf_max_↩ traceback_length > 1980

smtrat::SATModule< Settings >::UnorderedClauseLookup::UnorderedClauseHasher 2016

smtrat::UnsatCore< Solver, Strategy > 2016

smtrat::UnsatCore< Strategy, UnsatCoreStrategy::ModelExclusion > 2016

smtrat::unsatcore::UnsatCoreBackend< Solver, Strategy > 2017

smtrat::unsatcore::UnsatCoreBackend< Solver, UnsatCoreStrategy::ModelExclusion > 2018

smtrat::vs::unsignedTripleCmp 2018

smtrat::validation::ValidationPoint 2020

smtrat::validation::ValidationPrinter< SOF > 2021

smtrat::validation::ValidationSettings 2021

smtrat::lra::Value< T > 2022

smtrat::lra::Value< LRABoundType > 2022

smtrat::lra::Value< T1 > 2022

smtrat::lra::Variable< T1, T2 > 2030

smtrat::vb::Variable< T > 2040

smtrat::vb::Variable< ConstraintT > 2040

smtrat::vb::Variable< FormulaT > 2040

smtrat::lra::Variable< LRABoundType, LRAEntryType > 2030

smtrat::vb::VariableBounds< T > 2043

smtrat::vb::VariableBounds< ConstraintT > 2043

smtrat::vb::VariableBounds< FormulaT > 2043

smtrat::VariableCapsule 2050

smtrat::mcsat::variableordering::VariableIDs 2050

smtrat::VariableRewriteRule 2051

smtrat::mcsat::VariableSelector 2052

smtrat::VarSchedulerBase 2057

- smtrat::VarSchedulerMcsatBase 2068
  - smtrat::TheoryVarSchedulerStatic< vot > 1923
  - smtrat::VarSchedulerMcsatActivityPreferTheory< vot > 2064
  - smtrat::VarSchedulerMcsatBooleanFirst< vot > 2072
  - smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler > 2075
  - smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities > 2079
  - smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot > 2083
- smtrat::VarSchedulerMinisat 2087
  - smtrat::VarSchedulerFixedRandom 2061
- smtrat::VarSchedulerRandom 2090
- smtrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic > 2093

Minisat::vec< T > 2097

Minisat::vec< bool > 2097

Minisat::vec< char > 2097

Minisat::vec< double > 2097

Minisat::vec< Idx > 2097

Minisat::vec< int > 2097

Minisat::vec< Minisat::CRef > 2097

Minisat::vec< Minisat::lbool > 2097

Minisat::vec< Minisat::Lit > 2097

Minisat::vec< Minisat::Map::Pair > 2097

Minisat::vec< Minisat::vec< Minisat::Lit > > 2097

Minisat::vec< Minisat::vec< Minisat::Watcher > > 2097

Minisat::vec< std::pair< Abstraction *, Abstraction * > > 2097

Minisat::vec< unsigned > 2097

Minisat::vec< VarData > 2097

Minisat::vec< Vec > 2097

std::vector< T >

smtrat::cad::projection::Reducta< Poly > 1664

smtrat::parser::SExpressionSequence< T > 1733

smtrat::parser::conversion::VectorVariantConverter< Res > 2100

smtrat::parser::conversion::VectorVariantConverter< types::BVTerm > 2100

Minisat::Watcher 2132

smtrat::ICPModule< Settings >::weights 2133

benchmax::XMLWriter 2134

13 Data Structure Index

13.1 Data Structures

Here are the data structures with brief descriptions:

smtrat::AbstractModuleFactory 245

smtrat::parser::AbstractTheory
Base class for all theories 246

smtrat::cad::sample_compare::absvalue 249

smtrat::analyzer::AnalysisSettings 249

smtrat::analyzer::AnalyzerStatistics 249

smtrat::AnnotatedBVTerm 250

benchmax::slurm::ArchiveProperties
All properties needed to archive log files 251

smtrat::parser::ArithmeticTheory
Implements the theory of arithmetic, including LRA, LIA, NRA and NIA 251

smtrat::parser::types::ArithmeticTheory
Types of the arithmetic theory 254

smtrat::execution::Assertion 254

smtrat::cad::preprocessor::AssignmentCollector 255

smtrat::mcsat::arithmetic::AssignmentFinder 256

smtrat::mcsat::smtaf::AssignmentFinder< Settings > 256

smtrat::mcsat::arithmetic::AssignmentFinder_detail 256

smtrat::mcsat::smtaf::AssignmentFinder_SMT 257

smtrat::cadcells::datastructures::detail::AssignmentProperties 258

smtrat::parser::Attribute
Represents an Attribute 258

smtrat::parser::AttributeParser 259

smtrat::parser::AttributeValueParser 260

smtrat::AxiomFactory 260

benchmax::Backend
Base class for all backends 261

smtrat::Backend< Settings > 263

smtrat::BackendLink 266

smtrat::BackendSynchronisation 267

smtrat::Backtrackable< UnionFind > 267

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 269

smtrat::cad::BaseProjection< Settings > 271

smtrat::cad::BaseSettings 277

smtrat::expression::simplifier::BaseSimplifier 278

smtrat::cad::Origin::BaseType 280

smtrat::BEModule< Settings > 282

benchmax::BenchmarkResult
Results for a single benchmark run 311

benchmax::BenchmarkSet
A set of benchmarks from some common base directory 313

benchmax::settings::BenchmarkSettings
Settings for benchmarks 314

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 315

smtrat::expression::BinaryExpression 317

smtrat::datastructures::BinaryHeap< T > 318

smtrat::parser::BinaryParser
Parses binaries: `#b[01]+` 319

smtrat::parser::BitvectorTheory
Implements the theory of bitvectors 320

smtrat::parser::types::BitvectorTheory
Types of the theory of bitvectors 323

smtrat::BlastedConstr 323

smtrat::BlastedPoly 324

smtrat::mcsat::Bookkeeping
Represent the trail, i.e 325

smtrat::parser::BooleanEncodingTheory
Implements the theory of bitvectors 327

Minisat::BoolOption 329

smtrat::BoolUEQRewriter 331

smtrat::lra::Bound< T1, T2 >
Stores a bound, which could be an upper "<= b" or a lower bound ">= b" for a bound value b 331

smtrat::vb::Bound< T >
Class for the bound of a variable 342

smtrat::mcsat::fm::Bound 346

smtrat::Branching
Stores all necessary information of an branch, which can be used to detect probable infinite loop of branchings 347

smtrat::BVAnnotation 348

smtrat::BVDirectEncoder 350

smtrat::BVModule< Settings > 350

smtrat::by_address_hasher< IterType >
Hash an iterator by the pointed-to address 379

smtrat::cad::CAD< Settings > 380

smtrat::qe::cad::CAD< Settings > 383

smtrat::cad::CADConstraints< BT > 384

smtrat::cad::CADCore< CH > 388

smtrat::cad::CADCore< CoreHeuristic::BySample > 388

smtrat::cad::CADCore< CoreHeuristic::EnumerateAll > 389

smtrat::cad::CADCore< CoreHeuristic::Interleave > 389

smtrat::cad::CADCore< CoreHeuristic::PreferProjection > 390

smtrat::cad::CADCore< CoreHeuristic::PreferSampling > 390

smtrat::qe::cad::CADElimination 390

smtrat::cad::CADPreprocessor 391

smtrat::cad::CADPreprocessorSettings 392

smtrat::qe::cad::CADSettings 393

smtrat::CardinalityEncoder 394

smtrat::cadcells::representation::cell< H >
Note: If connected(i) holds, then the indexed root ordering must contain an ordering between the interval bounds 395

smtrat::cadcells::representation::cell< CellHeuristic::BIGGEST_CELL > 395

smtrat::cadcells::representation::cell< CellHeuristic::CHAIN_EQ > 396

smtrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS > 396

smtrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS_EQ > 396

smtrat::cadcells::operators::properties::cell_connected 397

smtrat::cadcells::datastructures::CellDescription
Holds the description of a cell 397

smtrat::cadcells::datastructures::CellRepresentation< P >
Represents a cell 399

delta::Checker
This class takes care of calling the actual solver 400

benchmax::slurm::ChunkedSubmitfileProperties
All properties needed to create a submit file 401

Minisat::Clause 403

Minisat::ClauseAllocator 406

smtrat::mcsat::ClauseChain
An explanation is either a single clause or a chain of clauses, satisfying the following properties: 408

smtrat::sat::detail::ClauseChecker 411

smtrat::CMakeOptionPrinter 411

Minisat::CMap< T > 411

smtrat::CNFerModule 413

smtrat::CoCoAGBModule< Settings > 444

smtrat::CollectBoolsInUEQs 474

smtrat::CollectionWithOrigins< Element, Origin > 475

smtrat::ICPModule< Settings >::comp
Typedefs: 477

smtrat::vs::Condition 477

benchmax::CondorBackend
Backend for the HTCondor batch system 480

smtrat::mcsat::fm::ConflictGenerator< Comparator > 482

smtrat::cad::ConflictGraph
Representation of a bipartite graph (C+S, E) of vertices C and S, representing the constraints and samples, respectively 483

smtrat::parser::Theories::ConstantAdder
Helper functor for addConstants() method 485

smtrat::cad::CADConstraints< BT >::ConstraintComparator 485

smtrat::cad::preprocessor::ConstraintUpdate 485

smtrat::ConstrTree 486

delta::Consumer
This class takes care of asynchronous execution of calls to the solver 486

smtrat::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 488

smtrat::icp::ContractionCandidate 489

smtrat::icp::contractionCandidateComp 494

smtrat::icp::ContractionCandidateManager 494

smtrat::parser::conversion::Converter< To > 496

smtrat::parser::conversion::Converter< FormulaT > 497

smtrat::parser::conversion::Converter< Poly > 497

smtrat::parser::conversion::Converter< types::BVTerm > 498

benchmax::settings::CoreSettings
Core settings 499

smtrat::settings::CoreSettings 500

smtrat::parser::CoreTheory
Implements the core theory of the booleans 501

smtrat::parser::types::CoreTheory
Types of the core theory 503

smtrat::cadcells::representation::covering< H > 504

smtrat::mcsat::arithmetic::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 504

smtrat::cadcells::representation::covering< CoveringHeuristic::CHAIN_COVERING > 505

smtrat::cadcells::representation::covering< CoveringHeuristic::DEFAULT_COVERING > 505

smtrat::mcsat::onecellcad::firstlevel::covering_at_level 506

smtrat::cadcells::datastructures::CoveringDescription
Describes a covering of the real line by CellDescriptions (given an appropriate sample) 506

smtrat::cadcells::datastructures::CoveringRepresentation< P >
Represents a covering over a cell 507

smtrat::mcsat::onecellcad::recursive::CoverNullification 508

smtrat::CSplitModule< Settings > 508

smtrat::CubeLIAModule< Settings > 538

smtrat::CurryModule< Settings > 568

smtrat::CycleEnumerator< FHG, Collector >
This class encapsulates an algorithm for enumerating all cycles 598

benchmax::Database
Dummy database that effectively disables storing to database. Set BENCHMAX_DATABASE to actually use a database 598

benchmax::DBAL 600

smtrat::parser::DecimalParser
Parses decimals: `numeral.0*numeral` 604

Minisat::DeepEqual< K > 604

Minisat::DeepHash< K > 604

smtrat::mcsat::fm::DefaultComparator
Does not order anything 605

smtrat::mcsat::vs::DefaultSettings 605

smtrat::mcsat::smtaf::DefaultSettings 605

smtrat::mcsat::fm::DefaultSettings 606

smtrat::cad::projection_compare::degree 606

smtrat::mcsat::onecellcad::recursive::DegreeAscending 606

smtrat::analyzer::DegreeCollector 606

smtrat::mcsat::onecellcad::recursive::DegreeDescending 607

smtrat::cadcells::datastructures::DelineatedDerivation< Properties >
A DelineatedDerivation is a BaseDerivation with a Delineation and an underlying SampledDerivation 607

smtrat::cadcells::datastructures::Delineation
Represents the delineation of a set of polynomials (at a sample), that is 609

smtrat::cadcells::representation::delineation< H > 611

smtrat::cadcells::representation::delineation< DelineationHeuristic::CHAIN > 611

smtrat::cadcells::datastructures::DelineationInterval
An interval of a delineation 611

smtrat::cadcells::datastructures::DelineationRepresentation< P >
Represents a delineation 612

smtrat::mcsat::icp::Dependencies 613

smtrat::cadcells::datastructures::DerivationRef< Properties >
A reference to a derivation, which is either 614

smtrat::mcsat::onecellcad::recursive::DontCoverNullification 616

smtrat::cad::debug::DotSubgraph 617

Minisat::DoubleOption 617

Minisat::DoubleRange 619

smtrat::expression::simplifier::DuplicateSimplifier 619

smtrat::DynamicPriorityQueue< T, Compare > 621

smtrat::dynarray< T >
Dynarray is similar to std::vector, but has only a reduced interface 623

smtrat::dynarray_allocator< T > 627

smtrat::datastructures::DynHeap< KeyType, Compare, WeightType > 627

smtrat::cad::ProjectionGlobalInformation::ECData 628

smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Edge
Internal type of an edge 629

smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >
Implements a list of edges 631

smtrat::ElementWithOrigins< Element, Origin > 633

smtrat::EMModule< Settings > 634

smtrat::EQGraph< VertexName > 665

smtrat::EQModule< Settings > 667

smtrat::EQPreprocessingModule< Settings > 697

Minisat::Equal< K > 726

delta::ErrorHandler 726

smtrat::parser::ErrorHandler 727

smtrat::ESModule< Settings > 727

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 758

smtrat::uf_impl::estimator< IteratorType, false > 758

smtrat::ExactlyOneCommanderEncoder 758

smtrat::execution::ExecutionState 760

smtrat::Executor< Strategy > 763

smtrat::mcsat::onecellcad::recursive::Explanation< Setting1, Setting2 > 770

smtrat::mcsat::vs::Explanation 771

smtrat::mcsat::fm::Explanation< Settings > 771

smtrat::mcsat::icp::Explanation 771

smtrat::mcsat::nlsat::Explanation 772

smtrat::mcsat::onecell::Explanation 772

smtrat::mcsat::onecellcad::levelwise::Explanation< Setting1, Setting2 > 773

smtrat::mcsat::vs::ExplanationGenerator< Settings > 773

smtrat::mcsat::nlsat::ExplanationGenerator 773

smtrat::expression::Expression 774

smtrat::expression::ExpressionContent 777

smtrat::expression::ExpressionConverter 778

smtrat::expression::ExpressionModifier 779

smtrat::expression::ExpressionPool 780

smtrat::parser::ParserState::ExpressionScope 782

smtrat::expression::ExpressionTypeChecker< T > 782

smtrat::expression::ExpressionVisitor 782

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 784

smtrat::mcsat::variableordering::detail::FeatureCollector< Objects >
This class manages features that are used to valuate variables on objects 784

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 785

smtrat::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 786

smtrat::impl::fixedsize_allocator_freelists 787

smtrat::impl::fixedsize_allocator_impl< T, SizeShift, LargeEnough, SmallEnough > 787

smtrat::impl::fixedsize_allocator_impl< T, SizeShift, true, true > 787

smtrat::impl::fixedsize_allocator_size_helper< T, Size > 789

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 789

smtrat::impl::fixedsize_freelists< SizeCurrent, SizeMax, LowEnough >
Publicly derive from all fixedsize_freelist types with chunk sizes between SizeCurrent and SizeMax 790

smtrat::impl::fixedsize_freelists< SizeCurrent, SizeMax, true > 791

smtrat::parser::FixedWidthConstant< T >
Represents a constant of a fixed width 791

smtrat::parseformula::FormulaCollector 792

smtrat::ICPModule< Settings >::formulaPtrComp 798

smtrat::FormulaWithOrigins
Stores a formula along with its origins 798

smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >
This class implements a forward hypergraph 801

smtrat::FouMoModule< Settings >
A module which applies the Fourier-Motzkin algorithm 804

smtrat::qe::fm::FourierMotzkinQE
Provides a simple implementation for Fourier Motzkin variable elimination for linear, existentially quantified constraints 834

smtrat::FPPModule< Settings > 835

smtrat::freelist< T >
Implements a fixed-(at-runtime)-size allocator based on a free list embedded in the space of currently unused objects 865

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 867

smtrat::cad::FullSampleComparator< Iterator, Strategy > 867

smtrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::T > 867

smtrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::Type > 867

smtrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::Value > 867

smtrat::parser::FunctionInstantiator 867

smtrat::GBModule< Settings >
A solver module based on Groebner basis 868

smtrat::GBModuleState< Settings >
A class to save the current state of a GBModule 906

smtrat::GBPPModule< Settings > 907

smtrat::EQModule< Settings >::not_asserted_equality::hash 938

Minisat::Hash< K > 938

std::hash< const smtrat::expression::ExpressionContent * > 938

std::hash< smtrat::expression::BinaryExpression > 938

std::hash< smtrat::expression::Expression > 939

std::hash< smtrat::expression::ITEExpression > 939

std::hash< smtrat::expression::NaryExpression > 939

std::hash< smtrat::expression::QuantifierExpression > 939

std::hash< smtrat::expression::UnaryExpression > 939

std::hash< smtrat::lra::Variable< T1, T2 > >
Implements std::hash for sort value 940

std::hash< smtrat::vs::Substitution > 940

std::hash< std::vector< T > > 940

std::hash_combiner 941

Minisat::Heap< Comp > 941

smtrat::parser::HexadecimalParser
Parses hexadecimals: `#x[0-9a-fA-F]+` 943

smtrat::icp::HistoryNode 944

smtrat::ICEModule< Settings > 948

smtrat::ICPModule< Settings > 978

smtrat::icp::IcpVariable 1009

smtrat::icp::icpVariableComp 1012

smtrat::parser::Identifier 1012

smtrat::parser::IdentifierParser 1014

smtrat::cad::debug::IDSanitizer 1014

smtrat::mcsat::fm::IgnoreCoreSettings 1014

smtrat::IncWidthModule< Settings > 1015

smtrat::mcsat::onecellcad::firstlevel::indexed_root 1044

smtrat::parser::IndexedFunctionInstantiator 1044

smtrat::cadcells::datastructures::IndexedRoot
Represents the i-th root of a multivariate polynomial at its main variable (given an appropriate sample) 1045

smtrat::cadcells::datastructures::IndexedRootOrdering
Describes an ordering of IndexedRoots 1046

smtrat::InequalitiesTable< Settings >
Datastructure for the GBModule 1047

smtrat::lra::Bound< T1, T2 >::Info
Stores some additional information for a bound 1050

smtrat::mcsat::InformationGetter 1052

smtrat::parser::Instantiator< V, T > 1053

smtrat::parser::InstructionHandler 1055

Minisat::Int64Range 1061

smtrat::IntBlastModule< Settings > 1062

smtrat::IntEqModule< Settings >
A module which checks whether the equations contained in the received (linear integer) formula have a solution 1091

smtrat::mcsat::icp::IntervalPropagation 1120

Minisat::IntOption 1121

Minisat::IntRange 1122

smtrat::is_sample_outside< S > 1123

smtrat::is_sample_outside< IsSampleOutsideAlgorithm::DEFAULT > 1123

smtrat::is_variant< T >
States whether a given type is a `boost::variant` 1123

smtrat::expression::ITEExpression 1123

benchmax::RandomizationAdaptor< T >::iterator 1124

smtrat::ModuleInput::IteratorCompare 1125

benchmax::Jobs
Represents a set of jobs, constructed as the cartesian product of a set of tools and a set of benchmarks 1125

smtrat::JunctorMerger 1126

smtrat::parser::KeywordParser
Parses keywords: `:simple_symbol` 1127

Minisat::lbool 1127

smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound 1128

smtrat::Module::Lemma 1130

Minisat::LessThan_default< T > 1131

smtrat::cad::sample_compare::level 1131

smtrat::cad::projection_compare::level 1131

smtrat::cad::ProjectionLevelInformation::LevelInfo 1131

smtrat::mcsat::onecellcad::levelwise::LevelwiseCAD 1133

smtrat::LevelWiseInformation< Settings > 1135

smtrat::cad::LiftingTree< Settings > 1138

smtrat::resource::Limiter 1141

smtrat::ICPModule< Settings >::linearVariable 1142

smtrat::mcsat::ClauseChain::Link 1142

Minisat::Lit 1143

benchmax::LocalBackend
This backend simply runs files sequentially on the local machine 1143

smtrat::LOG 1146

smtrat::parser::LogicParser 1146

smtrat::LongFormulaEncoder 1146

smtrat::LRAModule< Settings >
A module which performs the Simplex method on the linear part of it's received formula 1148

smtrat::LVEModule< Settings > 1183

smtrat::Manager
Base class for solvers 1213

Minisat::Map< K, D, H, E > 1223

benchmax::MathSAT
Tool wrapper for MathSAT for SMT-LIB 1225

smtrat::mcsat::fm::MaxSizeComparator
This heuristic chooses the explanation excluding the largest interval 1228

smtrat::MaxSMT< Solver, Strategy > 1228

smtrat::maxsmt::MaxSMTBackend< Solver, Strategy > 1229

smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::FU_MALIK_INCREMENTAL > 1229

smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::LINEAR_SEARCH > 1230

smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::MSU3 > 1230

smtrat::MCBModule< Settings > 1230

smtrat::mcsat::MCSATBackend< Settings > 1262

smtrat::mcsat::MCSATMixin< Settings > 1263

smtrat::expression::simplifier::MergeSimplifier 1270

benchmax::Minisat
Tool wrapper for a Minisat solver 1271

benchmax::Minisatp
Tool wrapper for the Minisatp solver for pseudo-Boolean problems 1274

smtrat::mcsat::fm::MinSizeComparator
This heuristic chooses the explanation excluding the smallest interval 1277

smtrat::mcsat::fm::MinVarCountComparator
This heuristic tries to minimize the number of variables occuring in the explanation 1277

smtrat::cad::MISGeneration< heuristic > 1277

smtrat::MixedSignEncoder 1279

smtrat::cad::ModelBasedProjection< incrementality, backtracking, Settings > 1280

smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >
This class implements a projection that supports no incrementality and expects backtracking to be in order 1286

smtrat::Module
A base class for all kind of theory solving methods 1293

smtrat::ModuleFactory< Module > 1323

smtrat::ModuleInput
The input formula a module has to consider for it's satisfiability check 1324

smtrat::settings::ModuleSettings 1330

smtrat::Module::ModuleStatistics 1331

smtrat::ModuleWrapper< M > 1332

smtrat::MonomialMappingByVariablePool 1333

smtrat::expression::NaryExpression 1333

smtrat::expression::simplifier::NegationSimplifier 1334

smtrat::NewCADModule< Settings > 1336

smtrat::NewCoveringModule< Settings > 1366

smtrat::NNFPreparation 1395

smtrat::NNFRewriter 1396

benchmax::ssh::Node
Specification of a compuation node for the SSH backend 1397

delta::Node
This class represents a node in a SMTLIB file 1398

delta::NodePrinter< pretty > 1402

smtrat::mcsat::onecellcad::recursive::NoHeuristic 1403

smtrat::NRAILModule< Settings > 1403

smtrat::parser::NumeralParser
Parses numerals: `(0 | [1-9][0-9]*)` 1433

smtrat::lra::Numeric 1433

smtrat::execution::Objective 1440

Minisat::OccLists< Idx, Vec, Deleted > 1440

smtrat::mcsat::onecellcad::OneCellCAD 1441

benchmax::settings::OperationSettings
Operation settings 1443

smtrat::Optimization< Solver > 1444

Minisat::Option 1445

Minisat::Option::OptionLt 1446

smtrat::datastructures::OrderPair< T1, T2, reversed > 1447

smtrat::datastructures::OrderPair< T1, T2, true > 1447

smtrat::cad::Origin
This class represents one or more origins of some object 1447

smtrat::cad::preprocessor::Origins 1450

smtrat::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 1451

Minisat::OutOfMemoryException 1452

smtrat::parser::OutputWrapper 1452

Minisat::Map< K, D, H, E >::Pair 1452

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 1453

smtrat::mcsat::ParallelExplanation< Backends > 1453

delta::Parser
This class parses a whole smtlib file into a hierarchy of Node objects 1453

smtrat::parser::ParserSettings 1454

smtrat::parser::ParserState 1455

smtrat::PBGaussModule< Settings > 1458

smtrat::PBPPModule< Settings > 1490

smtrat::PersistentUnionFind 1519

smtrat::PFEModule< Settings > 1520

smtrat::PModule 1550

smtrat::cadcells::operators::properties::poly_irreducible_sgn_inv 1581

smtrat::cadcells::operators::properties::poly_ord_inv 1581

smtrat::cadcells::operators::properties::poly_pdel 1582

smtrat::cadcells::operators::properties::poly_sgn_inv 1582

smtrat::cad::ProjectionPolynomialInformation::PolyInfo 1583

smtrat::cad::PolynomialComparator< PolynomialGetter > 1583

smtrat::cad::PolynomialLiftingQueue< PolynomialGetter > 1584

smtrat::cadcells::datastructures::PolyPool
A pool for polynomials 1585

smtrat::cadcells::datastructures::detail::PolyProperties 1586

smtrat::cadcells::datastructures::PolyRef
Refers to a polynomial 1587

smtrat::PolyTree 1587

smtrat::PolyTreeContent 1588

smtrat::PolyTreePool 1589

smtrat::cad::Preprocessor 1589

smtrat::cad::PreprocessorSettings 1591

benchmax::settings::PresetSettings 1591

smtrat::PriorityQueue< T, Compare > 1592

delta::Producer
This class iteratively applies the operators to a smtlib file until no further simplifications can be performed 1594

delta::ProgressBar
This class provides a simple way to show progress bars on the command line 1594

smtrat::cad::Projection< incrementality, backtracking, Settings > 1596

smtrat::qe::cad::Projection< Settings > 1601

smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings > 1608

smtrat::cad::Projection< Incrementality::FULL, BT, Settings > 1614

smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >
This class implements a projection that supports no incrementality and expects backtracking to be in order 1620

smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >
This class implements a projection that supports no incrementality and allows backtracking to be out of order 1627

smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings > 1633

smtrat::cad::projection_compare::ProjectionComparator< Strategy > 1639

smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::D > 1639

smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::ID > 1639

smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::LD > 1640

smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::PD > 1640

smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::SD > 1640

smtrat::cad::projection_compare::ProjectionComparator_impl< Args > 1641

smtrat::cad::ProjectionGlobalInformation 1641

smtrat::cad::ProjectionInformation 1642

smtrat::cad::ProjectionLevelInformation 1644

smtrat::cad::ProjectionOperator 1645

smtrat::cad::ProjectionPolynomialInformation 1645

smtrat::cadcells::datastructures::Projections
Encapsulates all computations on polynomials 1646

smtrat::cadcells::operators::PropertiesSet< Op > 1649

smtrat::cadcells::operators::PropertiesSet< op::mccallum > 1650

smtrat::cadcells::datastructures::PropertiesT< Ts >
Set of properties 1650

smtrat::cadcells::datastructures::PropertiesT< T, Ts... > 1650

smtrat::cadcells::datastructures::PropertiesTContent< T, is_flag > 1651

smtrat::cadcells::datastructures::PropertiesTContent< T, false > 1651

smtrat::cadcells::datastructures::PropertiesTContent< T, true > 1651

smtrat::cadcells::datastructures::property_hash< T > 1651

smtrat::PseudoBoolEncoder
Base class for a PseudoBoolean Encoder 1651

smtrat::PseudoBoolNormalizer 1653

smtrat::parser::QEParser 1653

smtrat::parser::QualifiedIdentifierParser 1654

smtrat::expression::QuantifierExpression 1655

smtrat::parser::QuantifierParser 1655

Minisat::Queue< T > 1656

smtrat::mcsat::icp::QueueEntry 1657

benchmax::RandomizationAdaptor< T >
Provides iteration over a given std::vector in a pseudo-randomized order 1657

smtrat::RationalCapsule 1658

smtrat::parser::RationalPolicies
Specialization of qi::real_policies for a Rational 1659

smtrat::fixedsize_allocator< T >::rebind< U > 1659

smtrat::mcsat::onecellcad::recursive::RecursiveCAD 1659

smtrat::cad::projection::Reducta< Poly >
Construct the set of reducta of the given polynomial 1664

Minisat::RegionAllocator< T > 1665

smtrat::NNFRewriter::remove_xor_first_arg 1667

smtrat::ReplaceVariablesRewriter 1667

delta::ErrorHandler::result< typename > 1668

smtrat::parser::ErrorHandler::result< typename > 1668

smtrat::cad::preprocessor::ResultantRule 1669

benchmax::Results
Stores results for for whole benchmax run 1669

smtrat::RNSEncoder 1670

smtrat::mcsat::onecellcad::firstlevel::root_indexer 1671

smtrat::cadcells::operators::properties::root_well_def 1672

smtrat::mcsat::arithmetic::RootIndexer 1673

smtrat::cad::Sample 1674

smtrat::cad::sample_compare::SampleComparator< Iterator, Strategy > 1676

smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LS > 1676

smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LT > 1676

smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LTA > 1677

smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LTS > 1677

smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LTSA > 1677

smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::S > 1677

smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::T > 1678

smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::TLSA > 1678

smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::TS > 1678

smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::TSA > 1679

smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Type > 1679

smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Value > 1679

smtrat::cad::sample_compare::SampleComparator_impl< It, Args > 1680

smtrat::cadcells::datastructures::SampledDerivation< Properties >
A SampledDerivation is a DelineatedDerivation with a sample and an DelineationInterval w.r.t 1680

smtrat::SampledDerivationRefCompare 1683

smtrat::cad::SampleIteratorQueue< Iterator, Comparator > 1683

smtrat::sampling< S > 1684

smtrat::sampling< SamplingAlgorithm::LOWER_UPPER_BETWEEN_SAMPLING > 1685

smtrat::SATModule< Settings >
Implements a module performing DPLL style SAT checking 1685

smtrat::parser::ScriptParser< Callee > 1721

smtrat::parser::ParserState::ScriptScope 1724

smtrat::mcsat::onecellcad::firstlevel::section 1724

smtrat::onecellcad::recursive::Section
Represent a cell's closed-interval-boundary object along one single axis by an irreducible, multivariate polynomial of level k 1725

smtrat::mcsat::onecellcad::Section
Represent a cell's (closed-interval-boundary) component along th k-th axis 1725

smtrat::mcsat::onecellcad::levelwise::SectionHeuristic1 1726

smtrat::mcsat::onecellcad::levelwise::SectionHeuristic2 1726

smtrat::mcsat::onecellcad::levelwise::SectionHeuristic3 1726

smtrat::mcsat::onecellcad::firstlevel::sector 1726

smtrat::mcsat::onecellcad::Sector
Represent a cell's open-interval boundary object along one single axis by two irreducible, multivariate polynomials of level k 1727

smtrat::onecellcad::recursive::Sector
Represent a cell's open-interval boundary object along one single axis by two irreducible, multivariate polynomials of level k 1728

smtrat::mcsat::onecellcad::levelwise::SectorHeuristic1 1728

smtrat::mcsat::onecellcad::levelwise::SectorHeuristic2 1729

smtrat::mcsat::onecellcad::levelwise::SectorHeuristic3 1729

smtrat::mcsat::SequentialAssignment< Backends > 1729

smtrat::mcsat::SequentialExplanation< Backends > 1729

delta::Settings
This class loads and checks the command line options with help of boost::program_options 1729

benchmax::settings::Settings
Generic class to manage runtime settings 1731

smtrat::settings::Settings 1732

smtrat::SettingsComponents 1732

benchmax::SettingsParser
Generic class to manage settings parsing using boost::program_options 1732

smtrat::SettingsParser 1732

smtrat::CNFerModule::SettingsType 1732

smtrat::parser::SExpressionParser 1732

smtrat::parser::SExpressionSequence< T > 1733

smtrat::ShortFormulaEncoder 1734

benchmax::simple_parser 1735

smtrat::parser::Theories::SimpleSortAdder
Helper functor for addSimpleSorts() method 1736

smtrat::parser::SimpleSymbolParser
Parses symbols: simple_symbol | quoted_symbol where 1736

smtrat::expression::simplifier::Simplifier 1737

smtrat::expression::simplifier::SimplifierChainCaller< chainID > 1737

smtrat::expression::simplifier::SimplifierChainCaller< 0 > 1737

smtrat::expression::simplifier::SingletonSimplifier 1738

smtrat::cad::sample_compare::size 1739

smtrat::parser::Skipper 1739

delta::Skipper
This class is a boost::spirit::qi grammar that matches whitespaces and smtlib comments 1740

benchmax::SlurmBackend
Backend for the Slurm workload manager 1740

benchmax::settings::SlurmBackendSettings
Settings for the Slurm backend 1743

smtrat::parser::SMTLIBParser 1744

benchmax::SMTRAT
Tool wrapper for SMT-RAT for SMT-LIB 1747

benchmax::SMTRAT_Analyzer
Tool wrapper for SMT-RAT for SMT-LIB 1750

benchmax::SMTRAT_OPB
Adapts the SMTRAT solver for OPB files 1753

smtrat::execution::SoftAssertion 1756

smtrat::settings::SolverSettings 1756

smtrat::parser::SortedVariableParser 1757

smtrat::parser::SortParser 1757

smtrat::parser::SpecConstantParser 1758

smtrat::SplitSOSModule< Settings > 1759

benchmax::SSHBackend 1789

benchmax::settings::SSHBackendSettings
Settings for SSH backend 1791

benchmax::ssh::SSHConnection
A wrapper class that manages a single SSH connection as specified in a Node object (with all its channels) 1792

smtrat::vs::State 1794

smtrat::StaticUnionFind 1820

smtrat::statistics::StatisticsSettings 1821

smtrat::StrategyGraph 1821

delta::String
This class can be used to create `std::string` objects using streaming operators 1822

Minisat::StringOption 1823

smtrat::parser::StringParser
Parses strings: `".+"` with escape sequences \\" and \\\\ 1824

smtrat::STropModule< Settings > 1824

benchmax::slurm::SubmitfileProperties
All properties needed to create a submit file 1854

smtrat::vs::Substitution 1856

smtrat::vs::substitutionPointerEqual 1860

smtrat::vs::substitutionPointerHash 1860

delta::SymbolParser
This class is a `boost::spirit::qi` grammar that matches all kinds of smtlib symbols, numbers etc 1860

smtrat::parser::SymbolParser 1861

smtrat::SymmetryModule< Settings > 1861

smtrat::lra::Tableau< Settings, T1, T2 > 1892

smtrat::lra::TableauEntry< T1, T2 > 1910

smtrat::mcsat::onecellcad::TagPoly
Tagged Polynomials 1914

smtrat::Task 1914

delta::TempFilenameGenerator
This class generates and reuses temporary filenames with a common prefix 1915

smtrat::parser::TermParser 1916

smtrat::mcsat::vs::helper::TestCandidate 1917

smtrat::parser::Theories
Combines all implemented theories and provides a single interface to interact with all theories at once 1918

smtrat::parser::TheoryError 1921

smtrat::mcsat::TheoryLevel 1921

smtrat::Module::TheoryPropagation 1922

smtrat::TheoryVarSchedulerStatic< vot >
Schedules theory variables statically 1923

smtrat::ThreadPool 1927

smtrat::cad::debug::TikzBasePrinter 1928

smtrat::cad::debug::TikzDAGPrinter 1929

smtrat::cad::debug::TikzHistoryPrinter 1930

smtrat::cad::debug::TikzTreePrinter 1931

benchmax::Tool
Base class for any tool 1933

benchmax::settings::ToolSettings
Tool-related settings 1937

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 1938

smtrat::TotalizerTree 1940

smtrat::cad::projection_compare::type 1940

smtrat::cad::sample_compare::type 1940

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 1941

smtrat::uf_impl::uf_data_wrapper< void, compression >
Special case of the uf_data_wrapper in case the data type is void 1942

smtrat::uf_impl::uf_rank_and_class< compression >
Wrapper class that contains the rank and parent class of some entry of the UF datastructure 1943

smtrat::uf_impl::uf_rank_and_class< true >
Compressed implementation of uf_rank_and_class 1944

smtrat::UFCegarModule< Settings > 1945

smtrat::UFRewriter 1974

smtrat::expression::UnaryExpression 1975

smtrat::cad::debug::UnifiedData 1975

smtrat::parser::types::UninterpretedTheory
Types of the theory of equalities and uninterpreted functions 1976

smtrat::parser::UninterpretedTheory
Implements the theory of equalities and uninterpreted functions 1977

smtrat::union_find< T, no_compression, MaxTracebackLength >
Implements a union-find datastructure, a datastructure containing an array of entries 1980

smtrat::UnionFindInterface< T, Implementation > 1984

smtrat::UnionFindModule< Settings > 1986

smtrat::SATModule< Settings >::UnorderedClauseLookup::UnorderedClauseHasher 2016

smtrat::UnsatCore< Solver, Strategy > 2016

smtrat::unsatcore::UnsatCoreBackend< Solver, Strategy > 2017

smtrat::unsatcore::UnsatCoreBackend< Solver, UnsatCoreStrategy::ModelExclusion >
Implements an easy strategy to obtain an unsatisfiable core 2018

smtrat::vs::unsignedTripleCmp 2018

smtrat::parser::UserFunctionInstantiator 2019

smtrat::validation::ValidationCollector 2020

smtrat::validation::ValidationPoint 2020

smtrat::validation::ValidationPrinter< SOF > 2021

smtrat::validation::ValidationSettings 2021

smtrat::lra::Value< T > 2022

smtrat::lra::Variable< T1, T2 > 2030

smtrat::vb::Variable< T >
Class for a variable 2040

smtrat::vb::VariableBounds< T >
Class to manage the bounds of a variable 2043

smtrat::VariableCapsule 2050

smtrat::mcsat::variableordering::VariableIDs 2050

smtrat::VariableRewriteRule 2051

smtrat::mcsat::VariableSelector 2052

smtrat::variant_hash_visitor 2053

smtrat::parser::conversion::VariantConverter< Res >
Converts variants to some type using the Converter class 2053

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` 2054

smtrat::parser::conversion::VariantVariantConverter< Res >
Converts variants to another variant type not using the Converter class 2057

smtrat::VarSchedulerBase
Base class for variable schedulers 2057

smtrat::VarSchedulerFixedRandom 2061

smtrat::VarSchedulerMcsatActivityPreferTheory< vot >
Activity-based scheduler preferring initially theory variables 2064

smtrat::VarSchedulerMcsatBase
Base class for all MCSAT variable scheduler 2068

smtrat::VarSchedulerMcsatBooleanFirst< vot >
Variable scheduling that all decides Boolean variables first before deciding any theory variable 2072

smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >
Variable scheduling that all decides theory variables first before deciding any Boolean variable 2075

smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >
Decides only Constraints occuring in clauses that are univariate in the current theory variable while the theory ordering is static 2079

smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >
Decides only Constraints univariate in the current theory variable while the theory ordering is static 2083

smtrat::VarSchedulerMinisat
Minisat's activity-based variable scheduling 2087

smtrat::VarSchedulerRandom
Random scheduler 2090

smtrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic >
Scheduler for SMT, implementing theory guided heuristics 2093

Minisat::vec< T > 2097

smtrat::parser::conversion::VectorVariantConverter< Res >
Converts a vector of variants to a vector of some type using the Converter class 2100

smtrat::VSModule< Settings > 2101

Minisat::Watcher
[Minisat related code] 2132

smtrat::ICPModule< Settings >::weights 2133

benchmax::XMLWriter
Writes results to a xml file 2134

benchmax::Z3
Tool wrapper for Z3 for SMT-LIB 2134

14 Namespace Documentation

14.1 benchmax Namespace Reference

Namespaces

- settings
- slurm
- ssh

Data Structures

- class Backend

  Base class for all backends.

- struct BenchmarkResult

  Results for a single benchmark run.

- class BenchmarkSet

  A set of benchmarks from some common base directory.

- class CondorBackend

  Backend for the HTCondor batch system.

- class Database

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

- class DBAL
- 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 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 RandomizationAdaptor

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

- class Results

  Stores results for for whole benchmax run.

- class SettingsParser

  Generic class to manage settings parsing using boost::program_options.

- class simple_parser
- class SlurmBackend

  Backend for the Slurm workload manager.

- 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 SSHBackend
- class Tool

  Base class for any tool.

- class XMLWriter

  Writes results to a xml file.

- 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)

14.1.1 Typedef Documentation

14.1.1.1 Job `using benchmax::Job = typedef std::pair<const Tool*, std::filesystem::path>`

Typedef for a job, consisting of a tool and an input file.

14.1.1.2 ToolPtr `using benchmax::ToolPtr = typedef std::unique_ptr<Tool>`

A std::unique_ptr to a Tool.

14.1.1.3 Tools `using benchmax::Tools = typedef std::vector<ToolPtr>`

A vector of ToolPtr.

14.1.2 Function Documentation

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

commandline	Program to execute.
stdout	Standard output.
print_to_stdout	Also print if true.

Returns Exit code of the program.

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.

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.

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.

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.

14.1.2.6 execute_backend()

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

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.

14.1.2.8 is_system_lib()

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

14.1.2.9 loadBenchmarks()

```
BenchmarkSet benchmax::loadBenchmarks ( )
```

Loads benchmark files from directory specified in settings.

14.1.2.10 loadTools() `Tools benchmax::loadTools ( )`

Load all tools from the tool settings.

14.1.2.11 logging_configure() `void benchmax::logging_configure ( ) [inline]`

Configure default logging for benchmax.

14.1.2.12 logging_quiet() `void benchmax::logging_quiet ( ) [inline]`

Configure quiet logging for benchmax.

14.1.2.13 logging_verbose() `void benchmax::logging_verbose ( ) [inline]`

Configure verbose logging for benchmax.

14.1.2.14 operator<<() [1/2]

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

Streaming operator for BenchmarkResult.

14.1.2.15 operator<<() [2/2]

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

Streaming operator for a generic tool.

14.1.2.16 parse_peak_memory()

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

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.

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

backend	Backend name.
tools	List of tools.
benchmarks	List of benchmarks.

14.1.2.19 settings_benchmarks() `const auto& benchmax::settings_benchmarks ( ) [inline]`

Return the benchmark settings.

14.1.2.20 settings_core() `const auto& benchmax::settings_core ( ) [inline]`

Retrieved core settings.

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.

14.1.2.22 settings_operation() `const auto& benchmax::settings_operation ( ) [inline]`

Retrieved operation settings.

14.1.2.23 settings_preset() `const auto& benchmax::settings_preset ( ) [inline]`

Return the Slurm settings.

14.1.2.24 settings_slurm() `const auto& benchmax::settings_slurm ( ) [inline]`

Return the Slurm settings.

14.1.2.25 settings_ssh() `const auto& benchmax::settings_ssh ( ) [inline]`

Return the SSH settings.

14.1.2.26 settings_tools() `const auto& benchmax::settings_tools ( ) [inline]`

Returns the tool settings.

14.2 benchmax::settings Namespace Reference

Data Structures

- struct BenchmarkSettings

  Settings for benchmarks.
- struct CoreSettings

  Core settings.
- struct OperationSettings

  Operation settings.
- struct PresetSettings
- struct Settings

  Generic class to manage runtime settings.
- struct SlurmBackendSettings

  Settings for the Slurm backend.
- struct SSHBackendSettings

  Settings for SSH backend.
- 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.

14.2.1 Function Documentation

14.2.1.1 finalize_benchmark_settings()

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

Postprocess benchmark settings.

14.2.1.2 finalize_preset_settings()

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

Postprocess preset settings.

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.

14.2.1.4 registerBenchmarkSettings()

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

Registers benchmark settings with the settings parser.

14.2.1.5 registerPresetSettings()

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

Registers preset settings with the settings parser.

14.2.1.6 registerSlurmBackendSettings()

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

Registers Slurm settings with the settings parser.

14.2.1.7 registerSSHBackendSettings()

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

Registers SSH settings with settings parser.

14.2.1.8 registerToolSettings()

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

Registers tool settings with the settings parser.

14.3 benchmax::slurm Namespace Reference

Data Structures

- struct ArchiveProperties

  All properties needed to archive log files.
- struct ChunkedSubmitfileProperties

  All properties needed to create a submit file.
- struct SubmitfileProperties

  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)

14.3.1 Function Documentation

14.3.1.1 archive_log_files()

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

Put all log files into an archive.

14.3.1.2 clear_log_files()

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

Clear log files from directory.

Parameters

basedir	Base directory to search in.

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

basedir	Base directory to search in.
jobid	ID of slurm job.

Returns List of result files.

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 )
```

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

p	Collection of all information needed.

Returns The filename of the submit file.

14.3.1.6 generate_submit_file_chunked()

```
std::string benchmax::slurm::generate_submit_file_chunked
(
            const ChunkedSubmitfileProperties & p )
```

14.3.1.7 is_job_finished()

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

Checks if the given job is finished.

Parameters

jobid	The job id.

Returns True if the job is completed.

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

output	Output of `sbatch`.

Returns Slurm job id.

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

content	Result information output.
name	Information to be parsed.

Returns Information specified in the output.

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

files	List of files.
remove	Boolean flag whether to actually remove files.

14.4 benchmax::ssh Namespace Reference

Data Structures

- struct Node

  Specification of a compuation 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).

14.5 carl Namespace Reference

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 ErrorHandler
- struct Node

  This class represents a node in a SMTLIB file.

- struct NodePrinter
- 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 ProgressBar

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

- class Settings

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

- struct Skipper

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

- class String

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

- struct SymbolParser

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

- class TempFilenameGenerator

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

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 BaseIteratorType
- typedef boost::spirit::line_pos_iterator< BaseIteratorType > 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))

14.6.1 Typedef Documentation

14.6.1.1 BaseIteratorType `typedef boost::spirit::istream_iterator delta::BaseIteratorType`

14.6.1.2 Iterator `typedef PositionIteratorType delta::Iterator`

14.6.1.3 NodeChangeSet `using delta::NodeChangeSet = typedef std::vector<Node>`

Set of new nodes.

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.

14.6.1.5 PositionIteratorType `typedef boost::spirit::line_pos_iterator<BaseIteratorType> delta::PositionIterato`

14.6.2 Function Documentation

14.6.2.1 BV_mergeShift() `NodeChangeSet delta::BV_mergeShift ( const Node & n )`

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

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

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

Node operator that returns all children of a node.

Parameters

n	Node.

Returns A set of replacements.

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

Node operator that provides meaningful replacements for variables.

Parameters

n	Node.

Returns A set of replacements.

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

Node operator that eliminated let expressions.

Parameters

n	Node.

Returns A set of replacements.

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

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

Parameters

n	Node.

Returns A set of replacements.

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

Node operator that provides meaningful replacements for numbers.

Parameters

n	Node.

Returns A set of replacements.

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

os	Output stream.
n	Node.

Returns `os`.

14.6.2.9 operator<<() [2/2]

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

14.6.2.10 reorderChildren()

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

Node operator that reorders children of a node.

Parameters

n	Node.

Returns A set of replacements.

14.7 Minisat Namespace Reference

Data Structures

- class BoolOption
- class Clause
- class ClauseAllocator
- class CMap
- struct DeepEqual
- struct DeepHash
- class DoubleOption
- struct DoubleRange
- struct Equal
- struct Hash
- class Heap
- struct Int64Range
- class IntOption
- struct IntRange
- class lbool

- struct LessThan_default
- struct Lit
- class Map
- class OccLists
- class Option
- class OutOfMemoryException
- class Queue
- class RegionAllocator
- class StringOption
- class vec
- struct Watcher

  [Minisat related code]

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

14.7.1 Typedef Documentation

14.7.1.1 CRef

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

14.7.1.2 Var

```
typedef int Minisat::Var
```

14.7.2 Function Documentation

14.7.2.1 append()

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

14.7.2.2 copy() [1/2]

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

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]
```

14.7.2.4 find()

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

14.7.2.5 hash() [1/4]

```
static uint32_t Minisat::hash (
    int32_t x ) [inline], [static]
```

14.7.2.6 hash() [2/4]

```
static uint32_t Minisat::hash (
    int64_t x ) [inline], [static]
```

14.7.2.7 hash() [3/4]

```
static uint32_t Minisat::hash (
    uint32_t x ) [inline], [static]
```

14.7.2.8 hash() [4/4]

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

14.7.2.9 mkLit()

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

14.7.2.10 neg()

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

14.7.2.11 operator<<() [1/5]

```
std::ostream& Minisat::operator<< (
    std::ostream & os,
    const Lit & l ) [inline]
```

14.7.2.12 operator<<() [2/5]

```
std::ostream& Minisat::operator<< (
    std::ostream & os,
    const Minisat::Clause & c ) [inline]
```

14.7.2.13 operator<<() [3/5]

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

14.7.2.14 operator<<() [4/5]

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

14.7.2.15 operator<<() [5/5]

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

14.7.2.16 operator^()

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

14.7.2.17 operator~()

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

14.7.2.18 printUsageAndExit()

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

14.7.2.19 remove()

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

14.7.2.20 selectionSort() [1/2]

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

14.7.2.21 selectionSort() [2/2]

```
template<class T , class LessThan >
void Minisat::selectionSort (
    T * array,
    int size,
    LessThan lt )
```

14.7.2.22 setHelpPrefixStr()

```
void Minisat::setHelpPrefixStr (
    const char * str )
```

14.7.2.23 setUsageHelp()

```
void Minisat::setUsageHelp (
    const char * str )
```

14.7.2.24 sign()

```
bool Minisat::sign (
    Lit p ) [inline]
```

14.7.2.25 sort() [1/4] template<class T >

```
static void Minisat::sort (
            T * array,
            int size )  [inline], [static]
```

14.7.2.26 sort() [2/4] template<class T , class LessThan >

```
void Minisat::sort (
            T * array,
            int size,
            LessThan lt )
```

14.7.2.27 sort() [3/4] template<class T >

```
void Minisat::sort (
            vec< T > & v )
```

14.7.2.28 sort() [4/4] template<class T , class LessThan >

```
void Minisat::sort (
            vec< T > & v,
            LessThan lt )
```

14.7.2.29 toInt() [1/3] int Minisat::toInt (

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

14.7.2.30 toInt() [2/3] int Minisat::toInt (

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

14.7.2.31 toInt() [3/3] int Minisat::toInt (

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

14.7.2.32 toLbool() lbool Minisat::toLbool (

```
            int v )  [inline]
```

14.7.2.33 toLit() Lit Minisat::toLit (

```
            int i )  [inline]
```

14.7.2.34 var() int Minisat::var (

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

14.7.2.35 xrealloc()

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

14.7.3 Variable Documentation

14.7.3.1 CONFLICT_CLAUSE

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

14.7.3.2 CRef_Lazy

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

14.7.3.3 CRef_TPropagation

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

14.7.3.4 CRef_Undef

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

14.7.3.5 LEMMA_CLAUSE

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

14.7.3.6 lit_Error

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

14.7.3.7 lit_Undef

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

14.7.3.8 NORMAL_CLAUSE

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

14.7.3.9 nprimes

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

14.7.3.10 PERMANENT_CLAUSE

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

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]
```

14.8 smtrat Namespace Reference

Functions for adding the delineation of a property.

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
- icp
- impl
- lra
- lve
- maxsmt

  Contains strategy implementations for max SMT computations.

- mcsat
- onecellcad
- options_detail
- parseformula
- parser
- preprocessor
- qe
- resource
- sat
- settings
- statistics
- uf_impl
- unsatcore

  Contains strategy implementations for unsat core computations.

- validation
- vb
- vs

Data Structures

- struct AbstractModuleFactory
- class AnnotatedBVTerm
- class AxiomFactory
- class Backend
- class BackendLink
- class BackendSynchronisation
- struct Backtrackable
- class BEModule
- class Bimap

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

- class BlastedConstr
- class BlastedPoly
- struct BoolUEQRewriter
- struct Branching

  Stores all necessary information of an branch, which can be used to detect probable infinite loop of branchings.

- class BVAnnotation
- class BVDirectEncoder
- class BVModule
- struct by_address_hasher

  Hash an iterator by the pointed-to address.

- class CardinalityEncoder

- struct CMakeOptionPrinter
- class CNFerModule
- class CoCoAGBModule
- struct CollectBoolsInUEQs
- class CollectionWithOrigins
- class ConstrTree
- class CSplitModule
- class CubeLIAModule
- class CurryModule
- struct CycleEnumerator

  This class encapsulates an algorithm for enumerating all cycles.

- class DynamicPriorityQueue
- class dynarray

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

- class dynarray_allocator
- class ElementWithOrigins
- class EMModule
- struct EQGraph
- class EQModule
- class EQPreprocessingModule
- class ESModule
- class ExactlyOneCommanderEncoder
- class Executor
- 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 fixedsize_freelist

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

- class FormulaWithOrigins

  Stores a formula along with its origins.

- class ForwardHyperGraph

  This class implements a forward hypergraph.

- class FouMoModule

  A module which applies the Fourier-Motzkin algorithm.

- class FPPModule
- class freelist

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

- class GBModule

  A solver module based on Groebner basis.

- class GBModuleState

  A class to save the current state of a GBModule.

- class GBPPModule
- class ICEModule
- class ICPModule
- class IncWidthModule
- class InequalitiesTable

  Datastructure for the GBModule.

- class IntBlastModule
- class IntEqModule

  A module which checks whether the equations contained in the received (linear integer) formula have a solution.

- struct is_sample_outside

- struct is_sample_outside< IsSampleOutsideAlgorithm::DEFAULT >
- 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`.

- struct JunctorMerger
- class LevelWiseInformation
- class LOG
- class LongFormulaEncoder
- class LRAModule

  A module which performs the Simplex method on the linear part of it's received formula.

- class LVEModule
- class Manager

  Base class for solvers.

- class MaxSMT
- class MCBModule
- class MixedSignEncoder
- class Module

  A base class for all kind of theory solving methods.

- struct ModuleFactory
- class ModuleInput

  The input formula a module has to consider for it's satisfiability check.

- class ModuleWrapper
- class MonomialMappingByVariablePool
- class NewCADModule
- class NewCoveringModule
- struct NNFPreparation
- class NNFRewriter
- class NRAILModule
- class Optimization
- 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 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.

- class PBGaussModule
- class PBPPModule
- struct PersistentUnionFind
- class PFEModule
- class PModule
- class PolyTree
- class PolyTreeContent
- class PolyTreePool
- class PriorityQueue
- class PseudoBoolEncoder

  Base class for a PseudoBoolean Encoder.

- class PseudoBoolNormalizer
- class RationalCapsule
- struct ReplaceVariablesRewriter
- class RNSEncoder
- struct SampledDerivationRefCompare
- struct sampling
- struct sampling< SamplingAlgorithm::LOWER_UPPER_BETWEEN_SAMPLING >
- class SATModule

Implements a module performing DPLL style SAT checking.

- class SettingsComponents
- class SettingsParser
- class ShortFormulaEncoder
- class SplitSOSModule
- struct StaticUnionFind
- class StrategyGraph
- class STropModule
- class SymmetryModule
- class Task
- class TheoryVarSchedulerStatic

  Schedules theory variables statically.

- class ThreadPool
- 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 UFCegarModule
- struct UFRewriter
- class union_find

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

- struct UnionFindInterface
- class UnionFindModule
- class UnsatCore
- class VariableCapsule
- class VariableRewriteRule
- struct variant_hash_visitor
- 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`.

- class VarSchedulerBase

  Base class for variable schedulers.

- class VarSchedulerFixedRandom
- class VarSchedulerMcsatActivityPreferTheory

  Activity-based scheduler preferring initially theory variables.

- class VarSchedulerMcsatBase

  Base class for all MCSAT variable scheduler.

- 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 VarSchedulerMcsatUnivariateClausesOnly

  Decides only Constraints occuring in clauses that are univariate in the current theory variable while the theory ordering is static.

- class VarSchedulerMcsatUnivariateConstraintsOnly

  Decides only Constraints univariate in the current theory variable while the theory ordering is static.

- class VarSchedulerMinisat

  Minisat's activity-based variable scheduling.

- class VarSchedulerRandom

  Random scheduler.

- class VarSchedulerSMTTheoryGuided

  Scheduler for SMT, implementing theory guided heuristics.

- class VSModule

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 EvalRationalMap = std::map< carl::Variable, Rational >
- using VarPolyInfo = carl::VariableInformation< true, Poly >
- 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 MaxSMTStrategy { MaxSMTStrategy::FU_MALIK_INCREMENTAL, MaxSMTStrategy::MSU3, MaxSMTStrategy::LINEAR_SEARCH }

- enum CoveringStatus { partial = 0, full = 1, unknown = 2, failed = 3 }
- enum SamplingAlgorithm { LOWER_UPPER_BETWEEN_SAMPLING }
- enum IsSampleOutsideAlgorithm { DEFAULT }
- enum TheoryGuidedDecisionHeuristicLevel : unsigned { TheoryGuidedDecisionHeuristicLevel::CONFLICT_FIRST, TheoryGuidedDecisionHeuristicLevel::SATISFIED_FIRST, TheoryGuidedDecisionHeuristicLevel::DISABLED }
- enum UnsatCoreStrategy { 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 &name)
- const auto & settings_validation ()
- void visiting (const Expression &expr)
- void testExpression ()
- std::ostream & operator<< (std::ostream &os, MaxSMTStrategy ucs)
- template<std::size_t N, class RandomAccessIterator , class Comparator >
  static void select_best (typename std::iterator_traits< RandomAccessIterator >::value_type *result, RandomAccessIterator begin, RandomAccessIterator end, Comparator &&compare)
- static void hash_combine (std::size_t &seed, std::size_t hash_value) noexcept
- carl::Sign sign_of (carl::Relation rel, bool invert)
- std::ostream & operator<< (std::ostream &os, const CoveringStatus &status)
- template<typename Settings >
  std::ostream & operator<< (std::ostream &os, const LevelWiseInformation< Settings > &levelWise↩Information)
- 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 (smtrat::VariableCapsule variableCapsule)
- FormulaT createTangentPlaneNEQOne (carl::Variable xVariable, carl::Variable yVariable, carl::Variable z↩ Variable, Rational aRational)
- FormulaT createTangentPlaneNEQTwo (carl::Variable xVariable, carl::Variable yVariable, carl::Variable z↩ Variable, Rational bRational)
- FormulaT createTangentPlaneNEQThree (carl::Variable xVariable, carl::Variable yVariable, carl::Variable z↩ Variable, Rational aRational, Rational bRational)
- FormulaT createTangentPlaneNEQFour (carl::Variable xVariable, carl::Variable yVariable, carl::Variable z↩ Variable, Rational aRational, Rational bRational)
- FormulasT createTangentPlaneNEQ (VariableCapsule variableCapsule, RationalCapsule rationalCapsule)
- FormulaT createTangentPlaneEQOne (carl::Variable xVariable, carl::Variable yVariable, carl::Variable z↩ Variable, Rational aRational)
- FormulaT createTangentPlaneEQTwo (carl::Variable xVariable, carl::Variable yVariable, carl::Variable z↩ Variable, Rational aRational)
- FormulasT createTangentPlaneEQ (VariableCapsule variableCapsule, RationalCapsule rationalCapsule)
- const smtrat::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 (smtrat::VariableCapsule variableCapsule, smtrat::VariableCapsule variableCapsuleInner)
- FormulaT createEquivalentToOriginalMonotonicityTwo (smtrat::VariableCapsule variableCapsule, smtrat::VariableCapsule variableCapsuleInner)
- FormulaT createEquivalentToOriginalMonotonicityThree (smtrat::VariableCapsule variableCapsule, smtrat::VariableCapsule variableCapsuleInner)
- RationalCapsule extractRationalCapsule (VariableCapsule variableCapsule, Model linearizedModel)
- FormulaT createOriginalMonotonicityOne (smtrat::VariableCapsule variableCapsule, smtrat::VariableCapsule variableCapsuleInner)
- FormulaT createOriginalMonotonicityTwo (smtrat::VariableCapsule variableCapsule, smtrat::VariableCapsule variableCapsuleInner)
- FormulaT createOriginalMonotonicityThree (smtrat::VariableCapsule variableCapsule, smtrat::VariableCapsule variableCapsuleInner)
- FormulasT createMonotonicity (VariableCapsule variableCapsuleOuter, VariableCapsule variableCapsule↩ Inner, Model absoluteValuedModel)
- FormulaT createCongruence (smtrat::VariableCapsule variableCapsuleOuter, smtrat::VariableCapsule variableCapsuleInner)
- 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)
- ModuleInput * originalFormula (new ModuleInput())
- carl::Variable createZVariable ()

  create variables and each time increment the suffix e.g., z_0, z_1, etc.

- carl::Monomial::Arg createSquareOfVariable (carl::Variable Variable)

  create a square of the variable

- void insertIntoMap (carl::Variable GeneratedVariable, carl::Monomial::Arg monomial)

  Insert the variable as key with monomial as value into the map The map is singeltone.
- int divisionOfExponentByTwo (int exponent)
- carl::Variable insertIntoMapOrRetrieveExistingVariable (carl::Monomial::Arg monomial)

  Check if the monomial is alreday inserted imto the map, If yes, returned the key which is a variable otherwise create new variable and insert the monomial into the map against this new variable.
- carl::Monomial::Arg abstractUnivariateMonomialForEvenExponent (carl::Variable Variable, int exponent)

  Replace square of variable (x_0^2) of a monomial (x_0^6) by a variable (z_0), create a new monomial (z_0^3) and insert this variable as key with monomial (x_0^2) as value in the map.
- carl::Variable abstractProductRecursively (std::list< carl::Variable > variables)

  It creates monomial by taking and popping first two items from the list, Insert this new monomial as value into the map against a newly generated variable, Each time the newly generated variable is insterted at the front of the list.
- carl::Monomial::Arg abstractUnivariateMonomialForOddExponent (carl::Variable Variable, int exponent)

  Replace square of variable (x_0^2) of a monomial (x_0^7) by a variable (z_0), create a linear monomial (z_1) and insert the variable as key with monomial as value in the map.
- carl::Monomial::Arg abstractUnivariateMonomial (carl::Variable Variable, int exponent)

  create monomial which is linear
- carl::Variable abstractMonomial (carl::Monomial::Arg monomial)

  It linearizes a non-linear monomial.
- FormulaT linearization (FormulaT formula)
- Answer toAnswer (int value)
- Answer isNRASatisfied (Model estimatedModel)

  Chekcs if the estimated Assignment satisfies the input NRA formula.
- Model createEstimatedAssignment (Model linearizedModel)

  Creates the estimated model.
- Model extendLinearizedModel (Model mModel, Model estimatedModel)

  create a model combining mModel, estimatedModel and rest of the variables are guessed.
- int rand (int min, int max)
- FormulasT unsatisfiedFormulas (AxiomFactory::AxiomType axiomType, FormulasT formulas, Model model, UNSATFormulaSelectionStrategy formulaSelectionStrategy)
- FormulasT refinement (AxiomFactory::AxiomType axiomType, Model linearizedModel, UNSATFormula↩ SelectionStrategy formulaSelectionStrategy)
- Rational factorial (Rational n)
- Rational factorial (std::size_t)
- std::string flatten_name (const carl::UTerm &term)
- std::string flatten_name_impl (const carl::UVariable &var)
- std::string flatten_name_impl (const carl::UFInstance &ufi)
- template<typename Instances >
  bool flattened_function_calls (const Instances &instances) noexcept
- 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()
- int zVariableCounter = 0
- std::string VariableName = "z_"
- static constexpr int COMMANDER_GROUP_SIZE = 4
- static const char * _cat = "CORE"
- static DoubleOption opt_var_decay (_cat, "var-decay", "The variable activity decay factor", 0.95, Double↩ Range(0, false, 1, false))

- static DoubleOption opt_clause_decay (_cat, "cla-decay", "The clause activity decay factor", 0.999, DoubleRange(0, false, 1, false))
- static DoubleOption opt_random_var_freq (_cat, "rnd-freq", "The frequency with which the decision heuristic tries to choose a random variable", 0, DoubleRange(0, true, 1, true))
- static DoubleOption opt_random_seed (_cat, "rnd-seed", "Used by the random variable selection", 91648253, DoubleRange(0, false, HUGE_VAL, false))
- static IntOption opt_ccmin_mode (_cat, "ccmin-mode", "Controls conflict clause minimization (0=none, 1=basic, 2=deep)", 2, IntRange(0, 2))
- static IntOption opt_phase_saving (_cat, "phase-saving", "Controls the level of phase saving (0=none, 1=limited, 2=full)", 2, IntRange(0, 2))
- static BoolOption opt_rnd_init_act (_cat, "rnd-init", "Randomize the initial activity", false)
- static BoolOption opt_luby_restart (_cat, "luby", "Use the Luby restart sequence", true)
- static IntOption opt_restart_first (_cat, "rfirst", "The base restart interval", 25, IntRange(1, INT32_MAX))
- static DoubleOption opt_restart_inc (_cat, "rinc", "Restart interval increase factor", 3, DoubleRange(1, false, HUGE_VAL, false))
- static DoubleOption opt_garbage_frac (_cat, "gc-frac", "The fraction of wasted memory allowed before a garbage collection is triggered", 0.20, DoubleRange(0, false, HUGE_VAL, false))

14.8.1 Detailed Description Functions for adding the delineation of a property.

File: VSModule.cpp.

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.

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

Heuristics for computing representations.

Implementation of derivation rules.

Contains all properties that are stored in a derivation.

Projection operators.

The delineation of a property is a set of indexed root expressions representing the critical points of the property.

E.g. for sign invariance of a polynomial, it is the set of roots of the polynomial.

Currently, only the McCallum based operator is implemented.

14.8.1.1 Relevant projection functions We refer to algorithms for usage examples.

14.8.1.1.1 Cells

- project_cell_properties
- project_basic_properties
- delineate_properties
- project_delineated_cell_properties

14.8.1.1.2 Coverings

- project_cell_properties (on each cell individually)
- project_basic_properties (on each cell individually)
- delineate_properties (on each cell individually)
- project_covering_properties

14.8.1.1.3 Delineaton

- project_basic_properties
- delineate_properties
- project_delineation_properties

Note that not all properties have a representation here as not all of them are stored but resolved directly in the 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.

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

Author Florian Corzilius corzilius@cs.rwth-aachen.de

Since 2010-05-11

Version 2014-11-27

14.8.2 Typedef Documentation

14.8.2.1 BacktrackableClasses

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

14.8.2.2 Classes

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

14.8.2.3 Conditionals

```
typedef std::vector< std::atomic_bool * > smtrat::Conditionals
```

A vector of atomic bool pointers.

14.8.2.4 ConditionEvaluation

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

14.8.2.5 ConditionFunction

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

14.8.2.6 ConstraintsT

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

14.8.2.7 ConstraintT

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

14.8.2.8 ContractionCandidates

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

14.8.2.9 DoubleInterval

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

14.8.2.10 EQClasses

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

14.8.2.11 EvalDoubleIntervalMap

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

14.8.2.12 EvalRationalIntervalMap

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

14.8.2.13 EvalRationalMap

`using smtrat::EvalRationalMap = typedef std::map<carl::Variable, Rational>`

14.8.2.14 Expression

`typedef expression::Expression smtrat::Expression`

14.8.2.15 Expressions

`typedef expression::Expressions smtrat::Expressions`

14.8.2.16 Factorization

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

14.8.2.17 Formula_Origins

`typedef std::map< FormulaT, std::shared_ptr< std::vector<FormulaT> > > smtrat::Formula_Origins`

14.8.2.18 FormulaOrigins

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

14.8.2.19 FormulaSetT

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

14.8.2.20 FormulasMultiT

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

14.8.2.21 FormulasT

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

14.8.2.22 FormulaT

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

14.8.2.23 InfinityValue

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

14.8.2.24 Integer

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

14.8.2.25 Model

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

14.8.2.26 ModelPolynomialSubstitution

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

14.8.2.27 ModelSubstitution

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

14.8.2.28 ModelValue

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

14.8.2.29 ModelVariable

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

14.8.2.30 MonomialMap

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

14.8.2.31 MonomialMapIterator

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

14.8.2.32 MultivariateRootT

```
using smtrat::MultivariateRootT = typedef carl::Multivariate↩
Root<Poly>
```

14.8.2.33 ObjectiveValues

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

14.8.2.34 Poly

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

14.8.2.35 Priority

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

14.8.2.36 Rational

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

14.8.2.37 RationalInterval

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

14.8.2.38 SingleFormulaOrigins

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

14.8.2.39 SqrtEx

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

14.8.2.40 StaticClasses

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

14.8.2.41 Statistics

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

14.8.2.42 TermT

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

14.8.2.43 thread_priority

`using smtrat::thread_priority = typedef std::pair<std::size_t, std::size_t>`

14.8.2.44 Type

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

14.8.2.45 VariableAssignmentT

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

14.8.2.46 VariableComparisonT

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

14.8.2.47 VariableUpperLower

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

14.8.2.48 VarPolyInfo

`using smtrat::VarPolyInfo = typedef carl::VariableInformation<true, Poly>`

14.8.3 Enumeration Type Documentation

14.8.3.1 Answer

`enum smtrat::Answer`

An enum with the possible answers a Module can give.

Enumerator

SAT	
UNSAT	
UNKNOWN	
ABORTED	
OPTIMAL	

14.8.3.2 CoveringStatus

`enum smtrat::CoveringStatus`

Enumerator

partial	
full	
unknown	
failed	

14.8.3.3 IsSampleOutsideAlgorithm `enum smtrat::IsSampleOutsideAlgorithm`

Enumerator

DEFAULT	

14.8.3.4 LemmaLevel `enum smtrat::LemmaLevel`

Enumerator

NONE	
NORMAL	
ADVANCED	

14.8.3.5 MaxSMTStrategy `enum smtrat::MaxSMTStrategy [strong]`

Enumerator

FU_MALIK_INCREMENTAL	
MSU3	
LINEAR_SEARCH	

14.8.3.6 SamplingAlgorithm `enum smtrat::SamplingAlgorithm`

Enumerator

LOWER_UPPER_BETWEEN_SAMPLING	

14.8.3.7 TheoryGuidedDecisionHeuristicLevel `enum smtrat::TheoryGuidedDecisionHeuristicLevel : unsigned [strong]`

Enumerator

CONFLICT_FIRST	
SATISFIED_FIRST	
DISABLED	

14.8.3.8 UnsatCoreStrategy `enum smtrat::UnsatCoreStrategy [strong]`

Enumerator

ModelExclusion	

14.8.4 Function Documentation

14.8.4.1 abEqualcCheck()

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

14.8.4.2 abGreatercCheck()

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

14.8.4.3 abLesscCheck()

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

14.8.4.4 abstractMonomial()

```
carl::Variable smtrat::abstractMonomial (
    carl::Monomial::Arg monomial )
```

It linearizes a non-linear monomial.

Parameters

monomial	a monomoal object

Returns a linear variable

14.8.4.5 abstractProductRecursively()

```
carl::Variable smtrat::abstractProductRecursively (
    std::list< carl::Variable > variables )
```

It creates monomial by taking and popping first two items from the list, Insert this new monomial as value into the map against a newly generated variable, Each time the newly generated variable is insterted at the front of the list.

Parameters

variables	List of carl variables

Returns created new linear variable

14.8.4.6 abstractUnivariateMonomial()

```
carl::Monomial::Arg smtrat::abstractUnivariateMonomial (
    carl::Variable Variable,
    int exponent )
```

create monomial which is linear

Parameters

Variable	a carl Variable object
exponent	even or odd exponent of the variable

Returns a linear monomial

14.8.4.7 abstractUnivariateMonomialForEvenExponent()

```
carl::Monomial::Arg smtrat::abstract↩
UnivariateMonomialForEvenExponent (
    carl::Variable Variable,
    int exponent )
```

Replace square of variable (x_0^2) of a monomial (x_0^6) by a variable (z_0), create a new monomial (z_0^3) and insert this variable as key with monomial (x_0^2) as value in the map.

Parameters

Variable	a carl::Variable object
exponent	an even exponent

Returns new monomial

14.8.4.8 abstractUnivariateMonomialForOddExponent()

```
carl::Monomial::Arg smtrat::abstract↩
UnivariateMonomialForOddExponent (
    carl::Variable Variable,
    int exponent )
```

Replace square of variable (x_0^2) of a monomial (x_0^7) by a variable (z_0), create a linear monomial (z_1) and insert the variable as key with monomial as value in the map.

Parameters

Variable	a carl::Variable object
exponent	an odd exponent

Returns new linear monomial

14.8.4.9 analyze_file()

```
int smtrat::analyze_file (
    const std::string & )
```

14.8.4.10 analyze_formula()

```
analyzer::AnalyzerStatistics & smtrat::analyze_formula (
    const FormulaT & f )
```

14.8.4.11 annotateFormula()

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

14.8.4.12 CMakeOptions()

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

14.8.4.13 convert_to_cnf_dimacs()

```
int smtrat::convert_to_cnf_dimacs (
    const std::string & ,
    const std::string & )
```

14.8.4.14 convert_to_cnf_smtlib()

```
int smtrat::convert_to_cnf_smtlib (
    const std::string & ,
    const std::string & )
```

14.8.4.15 createAbsoluteValuedModel()

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

14.8.4.16 createAuxiliaryVariable()

```
carl::Variable smtrat::createAuxiliaryVariable (
    carl::Variable variable )
```

create an auxiliary variable e.g.

aux_<Variable Name>

Parameters

variable	the variable for which auxiliary variable to be created

Returns created auxiliary variable

14.8.4.17 createCongruence()

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

14.8.4.18 createEquivalentToOriginalMonotonicityOne()

```
FormulaT smtrat::createEquivalentTo↩
OriginalMonotonicityOne (
    smtrat::VariableCapsule variableCapsule,
    smtrat::VariableCapsule variableCapsuleInner )
```

14.8.4.19 createEquivalentToOriginalMonotonicityThree()

```
FormulaT smtrat::createEquivalentTo↩
OriginalMonotonicityThree (
    smtrat::VariableCapsule variableCapsule,
    smtrat::VariableCapsule variableCapsuleInner )
```

14.8.4.20 createEquivalentToOriginalMonotonicityTwo()

```
FormulaT smtrat::createEquivalentTo↩
OriginalMonotonicityTwo (
    smtrat::VariableCapsule variableCapsule,
    smtrat::VariableCapsule variableCapsuleInner )
```

14.8.4.21 createEstimatedAssignment()

```
Model smtrat::createEstimatedAssignment (
    Model linearizedModel )
```

Creates the estimated model.

estimated model takes the solution from mModel or randomly chosen.

Parameters

linearizedModel	Model of linearized formula.

14.8.4.22 createICPGreater()

```
FormulasT smtrat::createICPGreater (
    VariableCapsule variableCapsule,
    RationalCapsule rationalCapsule )
```

14.8.4.23 createICPGreaterOne()

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

14.8.4.24 createICPGreaterTwo()

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

14.8.4.25 createICPLess()

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

14.8.4.26 createMonotonicity()

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

14.8.4.27 createOriginalMonotonicityOne()

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

14.8.4.28 createOriginalMonotonicityThree()

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

14.8.4.29 createOriginalMonotonicityTwo()

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

14.8.4.30 createSquareOfVariable()

```
carl::Monomial::Arg smtrat::createSquareOfVariable (
    carl::Variable Variable )
```

create a square of the variable

Parameters

Variable	a carl Variable object e.g., x_0

Returns a monomial object e.g., x_0^2

14.8.4.31 createTangentPlaneEQ()

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

14.8.4.32 createTangentPlaneEQOne()

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

14.8.4.33 createTangentPlaneEQTwo()

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

14.8.4.34 createTangentPlaneNEQ()

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

14.8.4.35 createTangentPlaneNEQFour()

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

14.8.4.36 createTangentPlaneNEQOne()

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

14.8.4.37 createTangentPlaneNEQThree()

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

14.8.4.38 createTangentPlaneNEQTwo()

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

14.8.4.39 createZero()

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

14.8.4.40 createZeroOne()

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

14.8.4.41 createZeroThree()

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

14.8.4.42 createZeroTwo()

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

14.8.4.43 createZVariable()

```
carl::Variable smtrat::createZVariable ( )
```

create variables and each time increment the suffix e.g., z_0, z_1, etc.

Returns a carl Variable object

14.8.4.44 divisionOfExponentByTwo()

```
int smtrat::divisionOfExponentByTwo (
    int exponent )
```

14.8.4.45 executeFile()

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

Parse the file and save it in formula.

Parameters

pathToInputFile	The path to the input smt2 file.
formula	A pointer to the formula object which holds the parsed input afterwards.
options	Save options from the smt2 file here.

14.8.4.46 extendLinearizedModel()

```
Model smtrat::extendLinearizedModel (
    Model mModel,
    Model estimatedModel )
```

create a model combining mModel, estimatedModel and rest of the variables are guessed.

This model checks the satifiability of the axioms.

Parameters

mModel	Model of linearized formula.
estimatedModel	the estimated model for original formula.

Returns the Abstract Model

14.8.4.47 extractRationalCapsule()

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

14.8.4.48 extractVariables()

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

14.8.4.49 factorial() [1/2]

```
Rational smtrat::factorial (
    Rational n )
```

14.8.4.50 factorial() [2/2]

```
Rational smtrat::factorial (
    std::size_t n )
```

14.8.4.51 flatten_name()

```
std::string smtrat::flatten_name (
    const carl::UTerm & term )
```

14.8.4.52 flatten_name_impl() [1/2]

```
std::string smtrat::flatten_name_impl (
    const carl::UFInstance & ufi )
```

14.8.4.53 flatten_name_impl() [2/2]

```
std::string smtrat::flatten_name_impl (
    const carl::UVariable & var )
```

14.8.4.54 flattened_function_calls()

```
template<typename Instances >
bool smtrat::flattened_function_calls (
            const Instances & instances )  [noexcept]
```

14.8.4.55 generateAbcPrimeForICP()

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

14.8.4.56 generateAbsFormula() [1/2]

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

| x1 | = (and (=> (x1 >= 0) (y1 = x1)) (=> (x1 < 0) (y1 = -x1)))

Parameters

variable	

Returns

14.8.4.57 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

variableLeft	
variableRight	
relation	

Returns

14.8.4.58 get_name() [1/2]

```
std::string smtrat::get_name (
    IsSampleOutsideAlgorithm samplingAlgorithm ) [inline]
```

14.8.4.59 get_name() [2/2]

```
std::string smtrat::get_name (
    SamplingAlgorithm samplingAlgorithm ) [inline]
```

14.8.4.60 handle_basic_options()

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

14.8.4.61 hash_combine()

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

14.8.4.62 insertIntoMap()

```
void smtrat::insertIntoMap (
    carl::Variable GeneratedVariable,
    carl::Monomial::Arg monomial )
```

Insert the variable as key with monomial as value into the map The map is singeltone.

Parameters

GeneratedVariable	
monomial	

14.8.4.63 insertIntoMapOrRetrieveExistingVariable()

```
carl::Variable smtrat::insertIntoMapOr↩
RetrieveExistingVariable (
            carl::Monomial::Arg monomial )
```

Check if the monomial is alreday inserted imto the map, If yes, returned the key which is a variable otherwise create new variable and insert the monomial into the map against this new variable.

Parameters

monomial	a monomoal object

Returns a carl::Variable object

14.8.4.64 is_sat()

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

14.8.4.65 isAnyRationalIsZero()

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

14.8.4.66 isCondition()

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

14.8.4.67 isNRASatisfied()

```
Answer smtrat::isNRASatisfied (
            Model estimatedModel )
```

Chekcs if the estimated Assignment satisfies the input NRA formula.

Parameters

estimatedModel	The estimated model.

14.8.4.68 linearization()

```
FormulaT smtrat::linearization (
            FormulaT formula )
```

14.8.4.69 operator<<() [1/10]

```
std::ostream& smtrat::operator<< (
    std::ostream & os,
    Answer a ) [inline]
```

14.8.4.70 operator<<() [2/10]

```
std::ostream & smtrat::operator<< (
    std::ostream & os,
    CMakeOptionPrinter cmop )
```

14.8.4.71 operator<<() [3/10]

```
std::ostream& smtrat::operator<< (
    std::ostream & os,
    const CoveringStatus & status ) [inline]
```

14.8.4.72 operator<<() [4/10]

```
template<typename Settings >
std::ostream& smtrat::operator<< (
    std::ostream & os,
    const LevelWiseInformation< Settings > & levelWiseInformation ) [inline]
```

14.8.4.73 operator<<() [5/10]

```
template<typename Module >
std::ostream& smtrat::operator<< (
    std::ostream & os,
    const ModuleFactory< Module > & mf ) [inline]
```

14.8.4.74 operator<<() [6/10]

```
std::ostream& smtrat::operator<< (
    std::ostream & os,
    const ModuleInput & mi ) [inline]
```

14.8.4.75 operator<<() [7/10]

```
template<typename TT , typename C >
std::ostream& smtrat::operator<< (
    std::ostream & os,
    const PriorityQueue< TT, C > & pq )
```

14.8.4.76 operator<<() [8/10]

```
std::ostream& smtrat::operator<< (
    std::ostream & os,
    MaxSMTStrategy ucs ) [inline]
```

14.8.4.77 operator<<() [9/10]

```
std::ostream& smtrat::operator<< (
    std::ostream & os,
    Module::LemmaType lt ) [inline]
```

14.8.4.78 operator<<() [10/10]

```
std::ostream& smtrat::operator<< (
    std::ostream & os,
    UnsatCoreStrategy ucs ) [inline]
```

14.8.4.79 originalFormula()

```
ModuleInput* smtrat::originalFormula (
    new ModuleInput() )
```

14.8.4.80 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.

14.8.4.81 parse_smtlib()

```
FormulaT smtrat::parse_smtlib (
    const std::string & )
```

14.8.4.82 parseInput()

```
template<typename Executor >
bool smtrat::parseInput (
    const std::string & pathToInputFile,
    Executor & e,
    bool & queueInstructions )
```

14.8.4.83 parseSMT2File()

```
bool smtrat::parseSMT2File (
    parser::InstructionHandler & handler,
    bool queueInstructions,
    std::istream & input )
```

14.8.4.84 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).

14.8.4.85 rand()

```
int smtrat::rand (
    int min,
    int max )
```

14.8.4.86 refinement()

```
FormulasT smtrat::refinement (
    AxiomFactory::AxiomType axiomType,
    Model linearizedModel,
    UNSATFormulaSelectionStrategy formulaSelectionStrategy )
```

14.8.4.87 run_dimacs_file()

```
template<typename Strategy >
int smtrat::run_dimacs_file (
            Strategy & ,
            const std::string &  )
```

14.8.4.88 run_opb_file()

```
template<typename Strategy >
int smtrat::run_opb_file (
            Strategy & ,
            const std::string &  )
```

14.8.4.89 select_best()

```
template<std::size_t N, class RandomAccessIterator , class Comparator >
static void smtrat::select_best (
            typename std::iterator_traits< RandomAccessIterator >::value_type * result,
            RandomAccessIterator begin,
            RandomAccessIterator end,
            Comparator && compare )  [static]
```

14.8.4.90 Settings()

```
const settings::Settings& smtrat::Settings ( )  [inline]
```

14.8.4.91 settings_analyzer()

```
const auto& smtrat::settings_analyzer ( )  [inline]
```

14.8.4.92 settings_core()

```
const auto& smtrat::settings_core ( )  [inline]
```

14.8.4.93 settings_module()

```
const auto& smtrat::settings_module ( )  [inline]
```

14.8.4.94 settings_parser()

```
const auto& smtrat::settings_parser ( )  [inline]
```

14.8.4.95 settings_solver()

```
const auto& smtrat::settings_solver ( )  [inline]
```

14.8.4.96 settings_statistics()

```
const auto& smtrat::settings_statistics ( )  [inline]
```

14.8.4.97 settings_validation()

```
const auto& smtrat::settings_validation ( )  [inline]
```

14.8.4.98 sign_of()

```
carl::Sign smtrat::sign_of (
            carl::Relation rel,
            bool invert )  [inline]
```

14.8.4.99 statistics_get()

```
template<typename T >
auto& smtrat::statistics_get (
            const std::string & name )
```

14.8.4.100 testExpression()

```
void smtrat::testExpression ( )
```

14.8.4.101 toAnswer()

```
Answer smtrat::toAnswer (
            int value )
```

14.8.4.102 unsatisfiedFormulas()

```
FormulasT smtrat::unsatisfiedFormulas (
            AxiomFactory::AxiomType axiomType,
            FormulasT formulas,
            Model model,
            UNSATFormulaSelectionStrategy formulaSelectionStrategy )
```

14.8.4.103 validation_get()

```
auto& smtrat::validation_get (
            const std::string & channel,
            const std::string & name )  [inline]
```

14.8.4.104 visiting()

```
void smtrat::visiting (
            const Expression & expr )
```

14.8.5 Variable Documentation

14.8.5.1 _cat

```
const char* smtrat::_cat = "CORE"  [static]
```

14.8.5.2 COMMANDER_GROUP_SIZE

```
constexpr int smtrat::COMMANDER_GROUP_SIZE = 4  [static], [constexpr]
```

14.8.5.3 EMPTY_MODEL

```
const Model smtrat::EMPTY_MODEL = Model()  [static]
```

14.8.5.4 opt_ccmin_mode

```
IntOption smtrat::opt_ccmin_mode(_cat, "ccmin-mode", "Controls conflict clause minimization (0=none, 1=basic, 2=deep)", 2, IntRange(0, 2))  [static]
```

14.8.5.5 opt_clause_decay

```
DoubleOption smtrat::opt_clause_decay(_cat, "cla-decay", "The clause activity decay factor", 0.999, DoubleRange(0, false, 1, false))  [static]
```

14.8.5.6 opt_garbage_frac

`DoubleOption smtrat::opt_garbage_frac(_cat, "gc-frac", "The fraction of wasted memory allowed before a garbage collection is triggered", 0.20, DoubleRange(0, false, HUGE_VAL, false)) [static]`

14.8.5.7 opt_luby_restart

`BoolOption smtrat::opt_luby_restart(_cat, "luby", "Use the Luby restart sequence", true) [static]`

14.8.5.8 opt_phase_saving

`IntOption smtrat::opt_phase_saving(_cat, "phase-saving", "Controls the level of phase saving (0=none, 1=limited, 2=full)", 2, IntRange(0, 2)) [static]`

14.8.5.9 opt_random_seed

`DoubleOption smtrat::opt_random_seed(_cat, "rnd-seed", "Used by the random variable selection", 91648253, DoubleRange(0, false, HUGE_VAL, false)) [static]`

14.8.5.10 opt_random_var_freq

`DoubleOption smtrat::opt_random_var_freq(_cat, "rnd-freq", "The frequency with which the decision heuristic tries to choose a random variable", 0, DoubleRange(0, true, 1, true)) [static]`

14.8.5.11 opt_restart_first

`IntOption smtrat::opt_restart_first(_cat, "rfirst", "The base restart interval", 25, IntRange(1, INT32_MAX)) [static]`

14.8.5.12 opt_restart_inc

`DoubleOption smtrat::opt_restart_inc(_cat, "rinc", "Restart interval increase factor", 3, DoubleRange(1, false, HUGE_VAL, false)) [static]`

14.8.5.13 opt_rnd_init_act

`BoolOption smtrat::opt_rnd_init_act(_cat, "rnd-init", "Randomize the initial activity", false) [static]`

14.8.5.14 opt_var_decay

`DoubleOption smtrat::opt_var_decay(_cat, "var-decay", "The variable activity decay factor", 0.95, DoubleRange(0, false, 1, false)) [static]`

14.8.5.15 VariableName

`std::string smtrat::VariableName = "z_"`

14.8.5.16 zVariableCounter

`int smtrat::zVariableCounter = 0`

14.9 smtrat::analyzer Namespace Reference

Data Structures

- struct AnalysisSettings
- struct AnalyzerStatistics
- struct DegreeCollector

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)

14.9.1 Function Documentation

14.9.1.1 analyze_cad_projections()

```
void smtrat::analyzer::analyze_cad_projections (
    const FormulaT & f,
    AnalyzerStatistics & stats )
```

14.9.1.2 analyze_cnf()

```
void smtrat::analyzer::analyze_cnf (
    const FormulaT & f,
    AnalyzerStatistics & stats )
```

14.9.1.3 analyze_formula_types()

```
void smtrat::analyzer::analyze_formula_types (
    const FormulaT & f,
    AnalyzerStatistics & stats )
```

14.9.1.4 analyze_variables()

```
void smtrat::analyzer::analyze_variables (
    const FormulaT & f,
    AnalyzerStatistics & stats )
```

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 )
```

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 )
```

14.9.1.7 registerAnalyzerSettings()

```
template<typename T >
void smtrat::analyzer::registerAnalyzerSettings (
    T & parser )
```

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 BaseProjection
- struct BaseSettings
- class CAD
- class CADConstraints
- struct CADCore
- struct CADCore< CoreHeuristic::BySample >
- struct CADCore< CoreHeuristic::EnumerateAll >
- struct CADCore< CoreHeuristic::Interleave >
- struct CADCore< CoreHeuristic::PreferProjection >
- struct CADCore< CoreHeuristic::PreferSampling >
- class CADPreprocessor
- struct CADPreprocessorSettings
- class ConflictGraph

  Representation of a bipartite graph (C+S, E) of vertices C and S, representing the constraints and samples, respectively.

- struct FullSampleComparator
- struct FullSampleComparator< Iterator, FullSampleCompareStrategy::T >
- struct FullSampleComparator< Iterator, FullSampleCompareStrategy::Type >
- struct FullSampleComparator< Iterator, FullSampleCompareStrategy::Value >
- class LiftingTree
- class MISGeneration
- class ModelBasedProjection
- class ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >

  This class implements a projection that supports no incrementality and expects backtracking to be in order.

- class Origin

  This class represents one or more origins of some object.

- struct PolynomialComparator
- class PolynomialLiftingQueue
- class Preprocessor
- struct PreprocessorSettings
- class Projection
- class Projection< Incrementality::FULL, Backtracking::HIDE, Settings >
- class Projection< Incrementality::FULL, BT, Settings >
- 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 ProjectionInformation
- class ProjectionLevelInformation
- struct ProjectionOperator
- class ProjectionPolynomialInformation
- class Sample
- class SampleIteratorQueue

Typedefs

- using ConstraintSelection = carl::Bitset
- using OptionalID = boost::optional< std::size_t >
- using RAN = carl::RealAlgebraicNumber< Rational >
- 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 Incrementality { Incrementality::NONE, Incrementality::SIMPLE, Incrementality::FULL }
- enum Backtracking { Backtracking::ORDERED, Backtracking::UNORDERED, Backtracking::HIDE }
- enum ProjectionType {
  ProjectionType::Collins, ProjectionType::Hong, ProjectionType::McCallum, ProjectionType::McCallum_partial, ProjectionType::Lazard, ProjectionType::Brown }
- enum ProjectionCompareStrategy {
  ProjectionCompareStrategy::D, ProjectionCompareStrategy::PD, ProjectionCompareStrategy::SD, ProjectionCompareStrategy
  ProjectionCompareStrategy::LD, ProjectionCompareStrategy::Default = ID }
- enum SampleCompareStrategy {
  SampleCompareStrategy::T, SampleCompareStrategy::TLSA, SampleCompareStrategy::TSA, SampleCompareStrategy::TS, SampleCompareStrategy::LT, SampleCompareStrategy::LTA, SampleCompareStrategy::LTS, SampleCompareStrategy::LTSA
  SampleCompareStrategy::LS, SampleCompareStrategy::S, SampleCompareStrategy::Type, SampleCompareStrategy::Value, SampleCompareStrategy::Default = LT }
- enum FullSampleCompareStrategy { FullSampleCompareStrategy::Type, FullSampleCompareStrategy::Value, FullSampleCompareStrategy::T, FullSampleCompareStrategy::Default = T }
- enum MISHeuristic {
  MISHeuristic::TRIVIAL, MISHeuristic::GREEDY, MISHeuristic::GREEDY_PRE, MISHeuristic::GREEDY_WEIGHTED, MISHeuristic::EXACT, MISHeuristic::HYBRID }
- enum CoreHeuristic {
  CoreHeuristic::BySample, CoreHeuristic::PreferProjection, CoreHeuristic::PreferSampling, CoreHeuristic::EnumerateAll, CoreHeuristic::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)

14.10.1 Typedef Documentation

14.10.1.1 Assignment

```
using smtrat::cad::Assignment = typedef std::map<carl::Variable, RAN>
```

14.10.1.2 ConstraintSelection

```
using smtrat::cad::ConstraintSelection = typedef carl::Bitset
```

14.10.1.3 ModelBasedProjectionT

```
template<typename Settings >
using smtrat::cad::ModelBasedProjectionT = typedef ModelBasedProjection<Settings::incrementality,
Settings::backtracking, Settings>
```

14.10.1.4 OptionalID

```
using smtrat::cad::OptionalID = typedef boost::optional<std::size_t>
```

14.10.1.5 ProjectionT

```
template<typename Settings >
using smtrat::cad::ProjectionT = typedef Projection<Settings::incrementality, Settings::backtracking,
Settings>
```

14.10.1.6 RAN

```
using smtrat::cad::RAN = typedef carl::RealAlgebraicNumber<Rational>
```

14.10.1.7 SampleLiftedWith

```
using smtrat::cad::SampleLiftedWith = typedef carl::Bitset
```

14.10.1.8 SampleRootOf

```
using smtrat::cad::SampleRootOf = typedef carl::Bitset
```

14.10.1.9 UPoly

```
using smtrat::cad::UPoly = typedef carl::UnivariatePolynomial<Poly>
```

14.10.2 Enumeration Type Documentation

14.10.2.1 Backtracking

```
enum smtrat::cad::Backtracking [strong]
```

Enumerator

ORDERED	
UNORDERED	
HIDE	

14.10.2.2 CoreHeuristic `enum smtrat::cad::CoreHeuristic [strong]`

Enumerator

BySample	
PreferProjection	
PreferSampling	
EnumerateAll	
Interleave	

14.10.2.3 FullSampleCompareStrategy `enum smtrat::cad::FullSampleCompareStrategy [strong]`

Enumerator

Type	
Value	
T	
Default	

14.10.2.4 Incrementality `enum smtrat::cad::Incrementality [strong]`

Enumerator

NONE	
SIMPLE	
FULL	

14.10.2.5 MISHeuristic `enum smtrat::cad::MISHeuristic [strong]`

Enumerator

TRIVIAL	
GREEDY	
GREEDY_PRE	
GREEDY_WEIGHTED	
EXACT	
HYBRID	

14.10.2.6 ProjectionCompareStrategy `enum smtrat::cad::ProjectionCompareStrategy [strong]`

Enumerator

D	
PD	
SD	
ID	
LD	
Default	

14.10.2.7 ProjectionType `enum smtrat::cad::ProjectionType [strong]`

Enumerator

Collins	
Hong	
McCallum	
McCallum_partial	
Lazard	
Brown	

14.10.2.8 SampleCompareStrategy `enum smtrat::cad::SampleCompareStrategy [strong]`

Enumerator

T	
TLSA	
TSA	
TS	
LT	
LTA	
LTS	
LTSA	
LS	
S	
Type	
Value	
Default	

14.10.3 Function Documentation

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 )
```

14.10.3.2 operator<<() [1/16]

```
template<Backtracking BT>
std::ostream& smtrat::cad::operator<< (
            std::ostream & os,
            const CADConstraints< BT > & cc )
```

14.10.3.3 operator<<() [2/16]

```
std::ostream& smtrat::cad::operator<< (
            std::ostream & os,
            const CADPreprocessor & cadpp ) [inline]
```

14.10.3.4 operator<<() [3/16]

```
std::ostream & smtrat::cad::operator<< (
            std::ostream & os,
            const ConflictGraph & cg )
```

14.10.3.5 operator<<() [4/16]

```
std::ostream& smtrat::cad::operator<< (
            std::ostream & os,
            const full::Polynomial & p ) [inline]
```

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
)
```

14.10.3.7 operator<<() [6/16]

```
template<typename PG >
std::ostream& smtrat::cad::operator<< (
            std::ostream & os,
            const PolynomialLiftingQueue< PG > & plq ) [inline]
```

14.10.3.8 operator<<() [7/16]

```
std::ostream& smtrat::cad::operator<< (
            std::ostream & os,
            const Preprocessor & cadpp ) [inline]
```

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 )
```

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 )
```

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 )
```

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 )
```

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 )
```

14.10.3.14 operator<<() [13/16]

```
std::ostream& smtrat::cad::operator<< (
            std::ostream & os,
            const ProjectionGlobalInformation & gi )  [inline]
```

14.10.3.15 operator<<() [14/16]

```
std::ostream& smtrat::cad::operator<< (
            std::ostream & os,
            const ProjectionLevelInformation::LevelInfo & li )  [inline]
```

14.10.3.16 operator<<() [15/16]

```
std::ostream& smtrat::cad::operator<< (
            std::ostream & os,
            const ProjectionPolynomialInformation::PolyInfo & pi )  [inline]
```

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]
```

14.10.3.18 settings_cadpp()

```
const auto & smtrat::cad::settings_cadpp ( )  [inline]
```

14.11 smtrat::cad::debug Namespace Reference

Data Structures

- struct DotSubgraph
- struct IDSanitizer
- class TikzBasePrinter
- class TikzDAGPrinter
- class TikzHistoryPrinter
- class TikzTreePrinter
- struct UnifiedData

14.12 smtrat::cad::full Namespace Reference

Typedefs

- using Polynomial = boost::optional< std::pair< UPoly, Origin > >

14.12.1 Typedef Documentation

14.12.1.1 Polynomial

```
using smtrat::cad::full::Polynomial = typedef boost::optional<std::pair<UPoly,Origin>
>
```

14.13 smtrat::cad::full_ec Namespace Reference

Typedefs

- using Polynomial = boost::optional< UPoly >

14.13.1 Typedef Documentation

14.13.1.1 Polynomial

```
using smtrat::cad::full_ec::Polynomial = typedef boost::optional<UPoly>
```

14.14 smtrat::cad::preprocessor Namespace Reference

Data Structures

- class AssignmentCollector
- struct ConstraintUpdate
- struct Origins
- class ResultantRule

Functions

- std::size_t complexity (const std::vector< FormulaT > &origin)

14.14.1 Function Documentation

14.14.1.1 complexity()

```
std::size_t smtrat::cad::preprocessor::complexity (
    const std::vector< FormulaT > & origin ) [inline]
```

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)

14.15.1 Function Documentation

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.

14.15.1.2 canBeRemoved()

```
bool smtrat::cad::projection::canBeRemoved (
    const UPoly & p )  [inline]
```

Checks whether a polynomial can safely be ignored.

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 is main variable. The given variable instead indicates the main variable of the resulting polynomial.

Parameters

variable	Main variable of the resulting polynomial.
p	Input polynomial.

Returns Discriminant of p in main variable variable.

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.

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.

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.

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

variable	Main variable of the resulting polynomial.
p	First input polynomial.
q	Second input polynomial.

Returns Resultant of p and q in main variable variable.

14.15.1.8 returnPoly()

```
template<typename Poly , typename Callback >
void smtrat::cad::projection::returnPoly (
            const Poly & p,
            Callback && cb )
```

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* $p \hookleftarrow$ *:* $\{leading_c oeff(p), discriminant(p)\}$.

- 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.*

14.16.1 Detailed Description

Contains the implementation of Browns projection operator as specified in [**?**] after Theorem 3.1.

14.16.2 Function Documentation

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.

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 $p \hookleftarrow$: $\{leading_c oeff(p), discriminant(p)\}$.

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* $p \hookleftarrow$ *:* $\bigcup_{B \in RED(p)} \{ldcf(B)\} \cup PSC(B, B')$.

- 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* $q \hookleftarrow$ *:* $\bigcup_{B_1 \in RED(p), B_2 \in RED(q)} PSC(B_1, B_2)$.

14.17.1 Detailed Description Contains the implementation of Collins projection operator as specified in [**?**] below Theorem 4.

14.17.2 Function Documentation

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↩ : $\bigcup_{B_1 \in RED(p), B_2 \in RED(q)} PSC(B_1, B_2)$.

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↩ : $\bigcup_{B \in RED(p)} \{ldcf(B)\} \cup PSC(B, B')$.

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↩ : $\bigcup_{F \in RED(p)} PSC(F, q)$.*

14.18.1 Detailed Description Contains the implementation of Hongs projection operator as specified in [**?**] in Section 2.2.

14.18.2 Function Documentation

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↩ : $\bigcup_{F \in RED(p)} PSC(F, q)$.

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.

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* $p \leftarrow : \{leading_c oeff(p), trailing_c oeff(p), discriminant(p)\}$.
- 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.*

14.19.1 Detailed Description

Contains the implementation of Lazards projection operator as specified in **[Lauard1994]**.

14.19.2 Function Documentation

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.

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 $\mathrm{p} \leftarrow : \{leading_c oeff(p), trailing_c oeff(p), discriminant(p)\}$.

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 $p \hookleftarrow : coefficients(p) \cup \{discriminant(p)\}$.

- 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* $q \hookleftarrow : \{resultant(p, q)\}$.

14.20.1 Detailed Description Contains the implementation of McCallums projection operator as specified in [?].

14.20.2 Function Documentation

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 $q \hookleftarrow : \{resultant(p, q)\}$.

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 $p \hookleftarrow : coefficients(p) \cup \{discriminant(p)\}$.

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)

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.

14.21.2 Function Documentation

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 )
```

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 )
```

14.22 smtrat::cad::projection_compare Namespace Reference

Data Structures

- struct degree
- struct level
- struct ProjectionComparator
- struct ProjectionComparator< ProjectionCompareStrategy::D >
- struct ProjectionComparator< ProjectionCompareStrategy::ID >
- struct ProjectionComparator< ProjectionCompareStrategy::LD >
- struct ProjectionComparator< ProjectionCompareStrategy::PD >
- struct ProjectionComparator< ProjectionCompareStrategy::SD >
- struct ProjectionComparator_impl
- struct type

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)

14.22.1 Typedef Documentation

14.22.1.1 Candidate

```
template<typename Poly >
using smtrat::cad::projection_compare::Candidate = typedef std::tuple<const Poly&, const Poly&,
std::size_t>
```

14.22.1.2 gt

```
using smtrat::cad::projection_compare::gt = typedef std::greater<>
```

14.22.1.3 lt

```
using smtrat::cad::projection_compare::lt = typedef std::less<>
```

14.22.2 Function Documentation

14.22.2.1 compare() [1/2]

```
template<typename Poly >
bool smtrat::cad::projection_compare::compare (
            const Candidate< Poly > & lhs,
            const Candidate< Poly > & rhs )
```

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 )
```

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.

14.22.2.4 get() [1/3]

```
template<typename Poly >
auto smtrat::cad::projection_compare::get (
            const Candidate< Poly > & c,
            degree  )
```

14.22.2.5 get() [2/3]

```
template<typename Poly >
auto smtrat::cad::projection_compare::get (
            const Candidate< Poly > & c,
            level  )
```

14.22.2.6 get() [3/3]

```
template<typename Poly >
auto smtrat::cad::projection_compare::get (
            const Candidate< Poly > & c,
            type  )
```

14.22.2.7 operator<<()

```
template<typename Poly >
std::ostream& smtrat::cad::projection_compare::operator<< (
            std::ostream & os,
            const Candidate< Poly > & c )
```

14.23 smtrat::cad::sample_compare Namespace Reference

Contains comparison operators for samples and associated helpers.

Data Structures

- struct absvalue
- struct level
- struct SampleComparator
- struct SampleComparator< Iterator, SampleCompareStrategy::LS >
- struct SampleComparator< Iterator, SampleCompareStrategy::LT >
- struct SampleComparator< Iterator, SampleCompareStrategy::LTA >
- struct SampleComparator< Iterator, SampleCompareStrategy::LTS >
- struct SampleComparator< Iterator, SampleCompareStrategy::LTSA >
- struct SampleComparator< Iterator, SampleCompareStrategy::S >
- struct SampleComparator< Iterator, SampleCompareStrategy::T >
- struct SampleComparator< Iterator, SampleCompareStrategy::TLSA >
- struct SampleComparator< Iterator, SampleCompareStrategy::TS >
- struct SampleComparator< Iterator, SampleCompareStrategy::TSA >
- struct SampleComparator< Iterator, SampleCompareStrategy::Type >
- struct SampleComparator< Iterator, SampleCompareStrategy::Value >
- struct SampleComparator_impl
- struct size
- struct type

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)

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`.

14.23.2 Typedef Documentation

14.23.2.1 gt

```
using smtrat::cad::sample_compare::gt = typedef std::greater<>
```

14.23.2.2 lt

```
using smtrat::cad::sample_compare::lt = typedef std::less<>
```

14.23.3 Function Documentation

14.23.3.1 compare() [1/2] template<typename It >

```
bool smtrat::cad::sample_compare::compare (
            const It & lhs,
            const It & rhs )
```

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 )
```

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.

14.23.3.4 get() [1/4] template<typename It >

```
auto smtrat::cad::sample_compare::get (
            const It & it,
            absvalue )
```

14.23.3.5 get() [2/4] template<typename It >

```
auto smtrat::cad::sample_compare::get (
            const It & it,
            level )
```

14.23.3.6 get() [3/4] template<typename It >

```
auto smtrat::cad::sample_compare::get (
            const It & it,
            size )
```

14.23.3.7 get() [4/4] template<typename It >

```
auto smtrat::cad::sample_compare::get (
            const It & it,
            type )
```

14.24 smtrat::cad::variable_ordering Namespace Reference

Functions

- std::vector< carl::Variable > triangular_ordering (const std::vector< Poly > &polys)

14.24.1 Function Documentation

14.24.1.1 triangular_ordering()

```
std::vector< carl::Variable > smtrat::cad::variable_ordering↩
::triangular_ordering (
            const std::vector< Poly > & polys )
```

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
- representation

Typedefs

- using RAN = carl::real_algebraic_number< Rational >
- using Assignment = carl::ran_assignment< Rational >
- using VariableOrdering = std::vector< carl::Variable >

Variables

- static const Assignment empty_assignment

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.

14.25.1.1 Quick start

For an introduction, we refer to the code of algorithms::onecell.

14.25.2 Typedef Documentation

14.25.2.1 Assignment

```
using smtrat::cadcells::Assignment = typedef carl::ran_assignment<Rational>
```

14.25.2.2 RAN

```
using smtrat::cadcells::RAN = typedef carl::real_algebraic_number<Rational>
```

14.25.2.3 VariableOrdering

`using smtrat::cadcells::VariableOrdering = typedef std::vector<carl↩::Variable>`

14.25.3 Variable Documentation

14.25.3.1 empty_assignment

`const Assignment smtrat::cadcells::empty_assignment [static]`

14.26 smtrat::cadcells::algorithms Namespace Reference

Various algorithms as well as helper functions for developing new algorithms.

Namespaces

- helper

Typedefs

- using PropSet = operators::PropertiesSet< op >::type

Functions

- template<cadcells::operators::op op, representation::DelineationHeuristic delineation_heuristic>
  std::optional< datastructures::SampledDerivationRef< typename operators::PropertiesSet< op >::type > > get_delineation (datastructures::Projections &proj, const FormulasT &cs, const Assignment &sample)

  Computes a delineation w.r.t.
- 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 FormulasT &constraints, const Assignment &sample)

  Computes an unsat covering w.r.t.
- std::optional< std::pair< FormulasT, FormulaT > > onecell (const FormulasT &constraints, const VariableOrdering &vars, const Assignment &sample)

  An MCAST-style single cell explanation function.
- template<cadcells::operators::op op>
  std::vector< datastructures::SampledDerivationRef< typename operators::PropertiesSet< op >::type > > get_unsat_intervals (const ConstraintT &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 VariableComparisonT &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 FormulaT &c, datastructures::Projections &proj, const Assignment &sample)

  Returns the unsat intervals of the given atom w.r.t.

Variables

- constexpr auto cell_heuristic = representation::LOWEST_DEGREE_BARRIERS
- constexpr auto covering_heuristic = representation::CHAIN_COVERING
- constexpr auto op = operators::op::mccallum
- constexpr bool use_delineation = false

14.26.1 Detailed Description

Various algorithms as well as helper functions for developing new algorithms.

14.26.2 Typedef Documentation

14.26.2.1 PropSet

`using smtrat::cadcells::algorithms::PropSet = typedef operators::PropertiesSet<op>↩::type`

14.26.3 Function Documentation

14.26.3.1 get_delineation()

```
template<cadcells::operators::op op, representation::Delineation↩
Heuristic delineation_heuristic>
std::optional<datastructures::SampledDerivationRef<typename operators::PropertiesSet<op>↩
::type> > smtrat::cadcells::algorithms::get_delineation (
            datastructures::Projections & proj,
            const FormulasT & cs,
            const Assignment & sample )
```

Computes a delineation w.r.t.

a set of constraints.

A set of polynomials that are univariate under a sample induces a delineation (i.e. the order of their roots, polynomials without roots and nullified polynomials). This delineation is generalized such that the ordering on the roots etc is invariant.

Parameters

constraints	Atoms of the same level.
sample	A sample such that all but the highest variable in constraints are assigned.

Returns A sampled derivation which contains the information to keep the delineation invariant.

14.26.3.2 get_level_covering()

```
template<cadcells::operators::op op, representation::Covering↩
Heuristic covering_heuristic>
std::optional<datastructures::SampledDerivationRef<typename operators::PropertiesSet<op>↩
::type> > smtrat::cadcells::algorithms::get_level_covering (
            datastructures::Projections & proj,
            const FormulasT & 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 unassinged 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

constraints	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.
sample	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.

14.26.3.3 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 ConstraintT & c,
            datastructures::Projections & proj,
            const Assignment & sample )
```

14.26.3.4 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 FormulaT & c,
            datastructures::Projections & proj,
            const Assignment & sample )
```

Returns the unsat intervals of the given atom w.r.t.

an underlying sample.

Parameters

c	A constraint or a variable comparison.
sample	A sample point such that the highest variable in c w.r.t. the variable odering in proj 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.

14.26.3.5 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 VariableComparisonT & c,
            datastructures::Projections & proj,
            const Assignment & sample )
```

14.26.3.6 onecell()

```
std::optional<std::pair<FormulasT, FormulaT> > smtrat::cadcells::algorithms↩
::onecell (
            const FormulasT & constraints,
            const VariableOrdering & vars,
            const Assignment & sample )
```

An MCAST-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 exists 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

constraints	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.
sample	A sample such that all but the highest variable in constraints are assigned.

Returns A pair of a set of constraints causign the conflict and a formula describing the cell.

14.26.4 Variable Documentation

14.26.4.1 cell_heuristic

```
constexpr auto smtrat::cadcells::algorithms::cell_heuristic = representation::LOWEST_DEG
[constexpr]
```

14.26.4.2 covering_heuristic

```
constexpr auto smtrat::cadcells::algorithms::covering_heuristic =
representation::CHAIN_COVERING [constexpr]
```

14.26.4.3 op

```
constexpr auto smtrat::cadcells::algorithms::op = operators::op::mccallum [constexpr]
```

14.26.4.4 use_delineation

```
constexpr bool smtrat::cadcells::algorithms::use_delineation = false
[constexpr]
```

14.27 smtrat::cadcells::algorithms::helper Namespace Reference

Functions

- MultivariateRootT as_multivariate_root (const datastructures::PolyPool &pool, carl::Variable main_var, datastructures::IndexedRoot r)

  Converts an datastructures::IndexedRoot to a MultivariateRootT.
- FormulaT to_formula (const datastructures::PolyPool &pool, carl::Variable main_var, datastructures::CellDescription &c)

  Converts a datastructures::CellDescription to a FormulaT.

14.27.1 Function Documentation

14.27.1.1 as_multivariate_root()

```
MultivariateRootT smtrat::cadcells::algorithms::helper::as_↩
multivariate_root (
            const datastructures::PolyPool & pool,
            carl::Variable main_var,
            datastructures::IndexedRoot r ) [inline]
```

Converts an datastructures::IndexedRoot to a MultivariateRootT.

14.27.1.2 to_formula()

```
FormulaT smtrat::cadcells::algorithms::helper::to_formula (
            const datastructures::PolyPool & pool,
            carl::Variable main_var,
            datastructures::CellDescription & c )
```

Converts a datastructures::CellDescription to a FormulaT.

14.28 smtrat::cadcells::datastructures Namespace Reference

Main datastructures.

Namespaces

- detail

Data Structures

- class BaseDerivation

  A BaseDerivation has a level and a set of properties of this level, and an underlying derivation representing the lower levels.

- class CellDescription

  Holds the description of a cell.

- struct CellRepresentation

  Represents a cell.

- class CoveringDescription

  Describes a covering of the real line by CellDescriptions (given an appropriate sample).

- struct CoveringRepresentation

  Represents a covering over a cell.

- class DelineatedDerivation

  A DelineatedDerivation is a BaseDerivation with a Delineation and an underlying SampledDerivation.

- class Delineation

  Represents the delineation of a set of polynomials (at a sample), that is.

- class DelineationInterval

  An interval of a delineation.

- struct DelineationRepresentation

  Represents a delineation.

- class DerivationRef

  A reference to a derivation, which is either.

- struct IndexedRoot

  Represents the i-th root of a multivariate polynomial at its main variable (given an appropriate sample).

- class IndexedRootOrdering

  Describes an ordering of IndexedRoots.

- class PolyPool

  A pool for polynomials.

- struct PolyRef

  Refers to a polynomial.

- class Projections

  Encapsulates all computations on polynomials.

- struct PropertiesT

  Set of properties.

- struct PropertiesT< T, Ts... >
- struct PropertiesTContent
- struct PropertiesTContent< T, false >
- struct PropertiesTContent< T, true >
- struct property_hash
- class SampledDerivation

  A SampledDerivation is a DelineatedDerivation with a sample and an DelineationInterval w.r.t.

Typedefs

- using RootMap = 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 >

Enumerations

- enum Bound { Bound::infty }

  Bound type.

Functions

- std::ostream & operator<< (std::ostream &os, const DelineationInterval &data)
- std::ostream & operator<< (std::ostream &os, const Delineation &data)
- bool lower_less (const DelineationInterval &del1, const DelineationInterval &del2)
- bool lower_equal (const DelineationInterval &del1, const DelineationInterval &del2)
- bool upper_less (const DelineationInterval &del1, const DelineationInterval &del2)
- 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)
- template<typename P >
  std::ostream & operator<< (std::ostream &os, const DelineationRepresentation< 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)
- std::ostream & operator<< (std::ostream &os, const CellDescription &data)
- std::ostream & operator<< (std::ostream &os, const CoveringDescription &data)
- std::ostream & operator<< (std::ostream &os, const IndexedRootOrdering &data)

14.28.1 Detailed Description Main datastructures.

14.28.1.1 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.

14.28.1.2 Basic datastructures The basic datastructures for representing mathematical objects are datastructures::IndexedRoot, datastructures::CellDescription, datastructures::CoveringDescription, and datastructures::IndexedRootO

14.28.1.3 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.

14.28.1.4 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.

14.28.1.5 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 underyling derivation). All operations are forwarded to lower levels whenever neccessary.

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.

14.28.1.6 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, datastructures::CoveringRepresentation or datastructures::DelineationRepresentation.

The heuristics for computing representations are in representation.

14.28.1.7 Operators The operators work on derivations and representations.

14.28.2 Typedef Documentation

14.28.2.1 BaseDerivationRef

```
template<typename Properties >
using smtrat::cadcells::datastructures::BaseDerivationRef = typedef std::shared_ptr<BaseDerivation<Properties
>
```

14.28.2.2 DelineatedDerivationRef

```
template<typename Properties >
using smtrat::cadcells::datastructures::DelineatedDerivationRef = typedef std::shared_ptr<DelineatedDerivation
>
```

14.28.2.3 PropertiesTSet

```
template<typename T >
using smtrat::cadcells::datastructures::PropertiesTSet = typedef boost::container::flat_set<T>
```

14.28.2.4 RootMap

```
using smtrat::cadcells::datastructures::RootMap = typedef std::map<RAN,
std::vector<IndexedRoot> >
```

14.28.2.5 SampledDerivationRef

```
template<typename Properties >
using smtrat::cadcells::datastructures::SampledDerivationRef = typedef std::shared_ptr<SampledDerivation<Prope
>
```

14.28.3 Enumeration Type Documentation

14.28.3.1 Bound

```
enum smtrat::cadcells::datastructures::Bound [strong]
```

Bound type.

Enumerator

infty	

14.28.4 Function Documentation

14.28.4.1 lower_equal()

```
bool smtrat::cadcells::datastructures::lower_equal (
    const DelineationInterval & del1,
    const DelineationInterval & del2 ) [inline]
```

14.28.4.2 lower_less()

```
bool smtrat::cadcells::datastructures::lower_less (
    const DelineationInterval & del1,
    const DelineationInterval & del2 ) [inline]
```

14.28.4.3 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.

14.28.4.4 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.

14.28.4.5 merge()

```
template<class T , class... Ts>
void smtrat::cadcells::datastructures::merge (
    PropertiesT< T, Ts... > & sets_a,
    const PropertiesT< T, Ts... > & sets_b )
```

14.28.4.6 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.

14.28.4.7 operator"!=() [1/2]

```
bool smtrat::cadcells::datastructures::operator!= (
    const IndexedRoot & lhs,
    const IndexedRoot & rhs ) [inline]
```

14.28.4.8 operator"!=() [2/2]

```
bool smtrat::cadcells::datastructures::operator!= (
    const PolyRef & lhs,
    const PolyRef & rhs ) [inline]
```

14.28.4.9 operator<() [1/2]

```
bool smtrat::cadcells::datastructures::operator< (
    const IndexedRoot & lhs,
    const IndexedRoot & rhs ) [inline]
```

14.28.4.10 operator<() [2/2]

```
bool smtrat::cadcells::datastructures::operator< (
    const PolyRef & lhs,
    const PolyRef & rhs ) [inline]
```

14.28.4.11 operator<<() [1/10]

```
std::ostream& smtrat::cadcells::datastructures::operator<< (
    std::ostream & os,
    const CellDescription & data ) [inline]
```

14.28.4.12 operator<<() [2/10]

```
template<typename P >
std::ostream& smtrat::cadcells::datastructures::operator<< (
    std::ostream & os,
    const CellRepresentation< P > & data ) [inline]
```

14.28.4.13 operator<<() [3/10]

```
std::ostream& smtrat::cadcells::datastructures::operator<< (
    std::ostream & os,
    const CoveringDescription & data ) [inline]
```

14.28.4.14 operator<<() [4/10]

```
template<typename P >
std::ostream& smtrat::cadcells::datastructures::operator<< (
    std::ostream & os,
    const CoveringRepresentation< P > & data )
```

14.28.4.15 operator<<() [5/10]

```
std::ostream& smtrat::cadcells::datastructures::operator<< (
    std::ostream & os,
    const Delineation & data ) [inline]
```

14.28.4.16 operator<<() [6/10]

```
std::ostream& smtrat::cadcells::datastructures::operator<< (
    std::ostream & os,
    const DelineationInterval & data ) [inline]
```

14.28.4.17 operator<<() [7/10]

```
template<typename P >
std::ostream& smtrat::cadcells::datastructures::operator<< (
    std::ostream & os,
    const DelineationRepresentation< P > & data )
```

14.28.4.18 operator<<() [8/10]

```
std::ostream& smtrat::cadcells::datastructures::operator<< (
    std::ostream & os,
    const IndexedRoot & data ) [inline]
```

14.28.4.19 operator<<() [9/10]

```
std::ostream& smtrat::cadcells::datastructures::operator<< (
    std::ostream & os,
    const IndexedRootOrdering & data ) [inline]
```

14.28.4.20 operator<<() [10/10]

```
std::ostream& smtrat::cadcells::datastructures::operator<< (
    std::ostream & os,
    const PolyRef & data ) [inline]
```

14.28.4.21 operator==() [1/2]

```
bool smtrat::cadcells::datastructures::operator== (
    const IndexedRoot & lhs,
    const IndexedRoot & rhs ) [inline]
```

14.28.4.22 operator==() [2/2]

```
bool smtrat::cadcells::datastructures::operator== (
    const PolyRef & lhs,
    const PolyRef & rhs ) [inline]
```

14.28.4.23 prop_get() [1/2]

```
template<class T , class... Ts>
const auto& smtrat::cadcells::datastructures::prop_get (
    const PropertiesT< T, Ts... > & sets )
```

14.28.4.24 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 )
```

14.28.4.25 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 )
```

14.28.4.26 prop_has() [2/2]

```
template<class T , class... Ts>
bool smtrat::cadcells::datastructures::prop_has (
    const PropertiesT< T, Ts... > & sets,
    const T & element )
```

14.28.4.27 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 )
```

14.28.4.28 prop_insert() [2/2]

```
template<class T , class... Ts>
void smtrat::cadcells::datastructures::prop_insert (
            PropertiesT< T, Ts... > & sets,
            const T & element )
```

14.28.4.29 upper_less()

```
bool smtrat::cadcells::datastructures::upper_less (
            const DelineationInterval & del1,
            const DelineationInterval & del2 ) [inline]
```

14.29 smtrat::cadcells::datastructures::detail Namespace Reference

Data Structures

- struct AssignmentProperties
- struct PolyProperties

14.30 smtrat::cadcells::helper Namespace Reference

Functions

- static size_t level_of (const VariableOrdering &order, const Poly &poly)

  Find the index of the highest variable (wrt.
- static carl::Variable main_var (const VariableOrdering &order, const Poly &poly)

14.30.1 Function Documentation

14.30.1.1 level_of()

```
static size_t smtrat::cadcells::helper::level_of (
            const VariableOrdering & order,
            const Poly & poly ) [static]
```

Find the index of the highest variable (wrt.

the ordering in 'order') that occurs with positive degree in 'poly'. We start counting at 1 and represent "no level/variable" as 0. Examples:

- level(2) == 0 wrt. any variable order
- level(0∗x+2) == 0 wrt. any variable order
- level(x+2) == 1 wrt. [x < y < z]
- level(x+2) == 2 wrt. [y < x < z]
- level(x+2) == 3 wrt. [y < z < x]
- level(x∗y+2) == 2 wrt. [x < y < z] because of y
- level(x∗y+2) == 2 wrt. [y < x < z] because of x
- level(x∗y+2) == 3 wrt. [x < z < y] because of y Preconditions:
- 'poly.gatherVariables()' must be a subset of 'order'.

14.30.1.2 main_var()

```
static carl::Variable smtrat::cadcells::helper::main_var (
            const VariableOrdering & order,
            const Poly & poly ) [static]
```

14.31 smtrat::cadcells::operators Namespace Reference

Namespaces

- delineation
- properties
- rules

Data Structures

- struct PropertiesSet
- struct PropertiesSet< op::mccallum >

Enumerations

- enum op { mccallum }

Functions

- std::string get_name (op op)
- std::ostream & operator<< (std::ostream &os, op op)
- template<op Op>
  void 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 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>
  void project_covering_properties (datastructures::CoveringRepresentation< typename PropertiesSet< Op >::type > &repr)

  Project covering properties.
- template<op Op>
  void project_delineation_properties (datastructures::DelineationRepresentation< typename PropertiesSet< Op >::type > &repr)

  Project delineation properties.
- template<>
  void 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 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<>
  void project_covering_properties< op::mccallum > (datastructures::CoveringRepresentation< PropertiesSet< op::mccallum >::type > &repr)
- template<>
  void project_delineation_properties< op::mccallum > (datastructures::DelineationRepresentation< PropertiesSet< op::mccallum >::type > &repr)

14.31.1 Enumeration Type Documentation

14.31.1.1 op `enum smtrat::cadcells::operators::op`

Enumerator

mccallum	

14.31.2 Function Documentation

14.31.2.1 delineate_properties() template<op Op>

```
void smtrat::cadcells::operators::delineate_properties (
            datastructures::DelineatedDerivation< typename PropertiesSet< Op >::type > &
deriv )  [inline]
```

Delineate properties.

14.31.2.2 delineate_properties< op::mccallum >() template<>

```
void smtrat::cadcells::operators::delineate_properties< op::mccallum > (
            datastructures::DelineatedDerivation< PropertiesSet< op::mccallum >::type > &
deriv )  [inline]
```

14.31.2.3 get_name() std::string smtrat::cadcells::operators::get_name (

```
            op op )  [inline]
```

14.31.2.4 operator<<() std::ostream& smtrat::cadcells::operators::operator<< (

```
            std::ostream & os,
            op op )  [inline]
```

14.31.2.5 project_basic_properties() template<op Op>

```
void 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.

14.31.2.6 project_basic_properties< op::mccallum >() template<>

```
void smtrat::cadcells::operators::project_basic_properties< op::mccallum > (
            datastructures::DelineatedDerivation< PropertiesSet< op::mccallum >::type > &
deriv )  [inline]
```

14.31.2.7 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 Mc↩ Callum).

14.31.2.8 project_cell_properties< op::mccallum >() template<>

```
bool smtrat::cadcells::operators::project_cell_properties< op::mccallum > (
            datastructures::SampledDerivation< PropertiesSet< op::mccallum >::type > & deriv
) [inline]
```

14.31.2.9 project_covering_properties() template<op Op>

```
void smtrat::cadcells::operators::project_covering_properties (
            datastructures::CoveringRepresentation< typename PropertiesSet< Op >::type > &
repr ) [inline]
```

Project covering properties.

14.31.2.10 project_covering_properties< op::mccallum >() template<>

```
void smtrat::cadcells::operators::project_covering_properties< op::mccallum > (
            datastructures::CoveringRepresentation< PropertiesSet< op::mccallum >::type > &
repr ) [inline]
```

14.31.2.11 project_delineated_cell_properties() template<op Op>

```
void 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.

14.31.2.12 project_delineated_cell_properties< op::mccallum >() template<>

```
void smtrat::cadcells::operators::project_delineated_cell_properties< op::mccallum > (
            datastructures::CellRepresentation< PropertiesSet< op::mccallum >::type > &
repr,
            bool cell_represents ) [inline]
```

14.31.2.13 project_delineation_properties() template<op Op>

```
void smtrat::cadcells::operators::project_delineation_properties (
            datastructures::DelineationRepresentation< typename PropertiesSet< Op >::type >
& repr ) [inline]
```

Project delineation properties.

14.31.2.14 project_delineation_properties< op::mccallum >() template<>

```
void smtrat::cadcells::operators::project_delineation_properties< op::mccallum > (
            datastructures::DelineationRepresentation< PropertiesSet< op::mccallum >::type >
& repr ) [inline]
```

14.32 smtrat::cadcells::operators::delineation Namespace Reference

Functions

- template<typename P >
  void delineate (datastructures::DelineatedDerivation< P > &deriv, const properties::poly_irreducible_sgn_inv &prop)

14.32.1 Function Documentation

14.32.1.1 delineate()

```
template<typename P >
void smtrat::cadcells::operators::delineation::delineate (
            datastructures::DelineatedDerivation< P > & deriv,
            const properties::poly_irreducible_sgn_inv & prop )
```

14.33 smtrat::cadcells::operators::properties Namespace Reference

Data Structures

- struct cell_connected
- struct poly_irreducible_sgn_inv
- struct poly_ord_inv
- struct poly_pdel
- struct poly_sgn_inv
- struct root_well_def

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_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)

14.33.1 Function Documentation

14.33.1.1 operator<() [1/6]

```
bool smtrat::cadcells::operators::properties::operator< (
            const cell_connected & lhs,
            const cell_connected & rhs ) [inline]
```

14.33.1.2 operator<() [2/6]

```
bool smtrat::cadcells::operators::properties::operator< (
            const poly_irreducible_sgn_inv & lhs,
            const poly_irreducible_sgn_inv & rhs ) [inline]
```

14.33.1.3 operator<() [3/6]

```
bool smtrat::cadcells::operators::properties::operator< (
            const poly_ord_inv & lhs,
            const poly_ord_inv & rhs ) [inline]
```

14.33.1.4 operator<() [4/6]

```
bool smtrat::cadcells::operators::properties::operator< (
            const poly_pdel & lhs,
            const poly_pdel & rhs ) [inline]
```

14.33.1.5 operator<() [5/6]

```
bool smtrat::cadcells::operators::properties::operator< (
    const poly_sgn_inv & lhs,
    const poly_sgn_inv & rhs ) [inline]
```

14.33.1.6 operator<() [6/6]

```
bool smtrat::cadcells::operators::properties::operator< (
    const root_well_def & lhs,
    const root_well_def & rhs ) [inline]
```

14.33.1.7 operator<<() [1/6]

```
std::ostream& smtrat::cadcells::operators::properties::operator<<
(
    std::ostream & os,
    const cell_connected & data ) [inline]
```

14.33.1.8 operator<<() [2/6]

```
std::ostream& smtrat::cadcells::operators::properties::operator<<
(
    std::ostream & os,
    const poly_irreducible_sgn_inv & data ) [inline]
```

14.33.1.9 operator<<() [3/6]

```
std::ostream& smtrat::cadcells::operators::properties::operator<<
(
    std::ostream & os,
    const poly_ord_inv & data ) [inline]
```

14.33.1.10 operator<<() [4/6]

```
std::ostream& smtrat::cadcells::operators::properties::operator<<
(
    std::ostream & os,
    const poly_pdel & data ) [inline]
```

14.33.1.11 operator<<() [5/6]

```
std::ostream& smtrat::cadcells::operators::properties::operator<<
(
    std::ostream & os,
    const poly_sgn_inv & data ) [inline]
```

14.33.1.12 operator<<() [6/6]

```
std::ostream& smtrat::cadcells::operators::properties::operator<<
(
    std::ostream & os,
    const root_well_def & data ) [inline]
```

14.33.1.13 operator==() [1/6]

```
bool smtrat::cadcells::operators::properties::operator== (
    const cell_connected & lhs,
    const cell_connected & rhs ) [inline]
```

14.33.1.14 operator==() [2/6]

```
bool smtrat::cadcells::operators::properties::operator== (
    const poly_irreducible_sgn_inv & lhs,
    const poly_irreducible_sgn_inv & rhs ) [inline]
```

14.33.1.15 operator==() [3/6]

```
bool smtrat::cadcells::operators::properties::operator== (
    const poly_ord_inv & lhs,
    const poly_ord_inv & rhs ) [inline]
```

14.33.1.16 operator==() [4/6]

```
bool smtrat::cadcells::operators::properties::operator== (
    const poly_pdel & lhs,
    const poly_pdel & rhs ) [inline]
```

14.33.1.17 operator==() [5/6]

```
bool smtrat::cadcells::operators::properties::operator== (
    const poly_sgn_inv & lhs,
    const poly_sgn_inv & rhs ) [inline]
```

14.33.1.18 operator==() [6/6]

```
bool smtrat::cadcells::operators::properties::operator== (
    const root_well_def & lhs,
    const root_well_def & rhs ) [inline]
```

14.34 smtrat::cadcells::operators::rules Namespace Reference

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_irrecubile_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_sgn_inv (datastructures::DelineatedDerivation< P > &deriv, datastructures::PolyRef poly)
- template<typename P>
  void poly_irrecubile_nonzero_sgn_inv (datastructures::DelineatedDerivation< P > &deriv, datastructures::PolyRef poly)
- template<typename P>
  void cell_connected (datastructures::SampledDerivation< P > &deriv, const datastructures::CellDescription &cell, const datastructures::IndexedRootOrdering &ordering)
- template<typename P>
  void cell_analytic_submanifold (datastructures::SampledDerivation< P > &deriv, const datastructures::CellDescription &)
- template<typename P>
  void poly_irreducible_sgn_inv_ec (datastructures::SampledDerivation< P > &deriv, const datastructures::CellDescription &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::CellDescription &cell)
- template<typename P >
  void cell_well_def (datastructures::SampledDerivation< P > &deriv, const datastructures::CellDescription &cell)
- template<typename P >
  void root_ordering_holds (datastructures::SampledDerivation< P > &deriv, const datastructures::IndexedRootOrdering &ordering)
- template<typename P >
  void poly_irreducible_sgn_inv (datastructures::SampledDerivation< P > &deriv, const datastructures::CellDescription &cell, const datastructures::IndexedRootOrdering &ordering, datastructures::PolyRef poly)
- template<typename P >
  void poly_irreducible_sgn_inv (datastructures::DelineatedDerivation< P > &deriv, const datastructures::IndexedRootOrdering &, datastructures::PolyRef poly)
- template<typename P >
  void covering_holds (datastructures::DelineatedDerivation< P > &deriv, const datastructures::CoveringDescription &covering, const datastructures::IndexedRootOrdering &ordering)

14.34.1 Function Documentation

14.34.1.1 cell_analytic_submanifold()

```
template<typename P >
void smtrat::cadcells::operators::rules::cell_analytic_submanifold (
            datastructures::SampledDerivation< P > & deriv,
            const datastructures::CellDescription &  )
```

14.34.1.2 cell_connected()

```
template<typename P >
void smtrat::cadcells::operators::rules::cell_connected (
            datastructures::SampledDerivation< P > & deriv,
            const datastructures::CellDescription & cell,
            const datastructures::IndexedRootOrdering & ordering )
```

14.34.1.3 cell_represents()

```
template<typename P >
void smtrat::cadcells::operators::rules::cell_represents (
            datastructures::SampledDerivation< P > & deriv,
            const datastructures::CellDescription & cell )
```

14.34.1.4 cell_well_def()

```
template<typename P >
void smtrat::cadcells::operators::rules::cell_well_def (
            datastructures::SampledDerivation< P > & deriv,
            const datastructures::CellDescription & cell )
```

14.34.1.5 covering_holds()

```
template<typename P >
void smtrat::cadcells::operators::rules::covering_holds (
            datastructures::DelineatedDerivation< P > & deriv,
            const datastructures::CoveringDescription & covering,
            const datastructures::IndexedRootOrdering & ordering )
```

14.34.1.6 is_trivial_root_well_def()

```
template<typename P >
bool smtrat::cadcells::operators::rules::is_trivial_root_well_def (
            datastructures::SampledDerivation< P > & deriv,
            datastructures::IndexedRoot root )
```

14.34.1.7 poly_irrecubile_nonzero_sgn_inv()

```
template<typename P >
void smtrat::cadcells::operators::rules::poly_irrecubile_nonzero_sgn_inv (
            datastructures::DelineatedDerivation< P > & deriv,
            datastructures::PolyRef poly )
```

14.34.1.8 poly_irrecubile_ord_inv()

```
template<typename P >
void smtrat::cadcells::operators::rules::poly_irrecubile_ord_inv (
            datastructures::SampledDerivation< P > & deriv,
            datastructures::PolyRef poly )
```

14.34.1.9 poly_irreducible_sgn_inv() [1/2]

```
template<typename P >
void smtrat::cadcells::operators::rules::poly_irreducible_sgn_inv (
            datastructures::DelineatedDerivation< P > & deriv,
            const datastructures::IndexedRootOrdering & ,
            datastructures::PolyRef poly )
```

14.34.1.10 poly_irreducible_sgn_inv() [2/2]

```
template<typename P >
void smtrat::cadcells::operators::rules::poly_irreducible_sgn_inv (
            datastructures::SampledDerivation< P > & deriv,
            const datastructures::CellDescription & cell,
            const datastructures::IndexedRootOrdering & ordering,
            datastructures::PolyRef poly )
```

14.34.1.11 poly_irreducible_sgn_inv_ec()

```
template<typename P >
void smtrat::cadcells::operators::rules::poly_irreducible_sgn_inv_ec (
            datastructures::SampledDerivation< P > & deriv,
            const datastructures::CellDescription & cell,
            datastructures::PolyRef poly )
```

14.34.1.12 poly_non_null()

```
template<typename P >
bool smtrat::cadcells::operators::rules::poly_non_null (
            datastructures::SampledDerivation< P > & deriv,
            datastructures::PolyRef poly )
```

14.34.1.13 poly_ord_inv()

```
template<typename P >
void smtrat::cadcells::operators::rules::poly_ord_inv (
            datastructures::SampledDerivation< P > & deriv,
            datastructures::PolyRef poly )
```

14.34.1.14 poly_pdel()

```
template<typename P >
bool smtrat::cadcells::operators::rules::poly_pdel (
            datastructures::SampledDerivation< P > & deriv,
            datastructures::PolyRef poly )
```

14.34.1.15 poly_sgn_inv()

```
template<typename P >
void smtrat::cadcells::operators::rules::poly_sgn_inv (
            datastructures::DelineatedDerivation< P > & deriv,
            datastructures::PolyRef poly )
```

14.34.1.16 root_ordering_holds()

```
template<typename P >
void smtrat::cadcells::operators::rules::root_ordering_holds (
            datastructures::SampledDerivation< P > & deriv,
            const datastructures::IndexedRootOrdering & ordering )
```

14.34.1.17 root_represents()

```
template<typename P >
void smtrat::cadcells::operators::rules::root_represents (
            datastructures::SampledDerivation< P > & deriv,
            datastructures::IndexedRoot root )
```

14.34.1.18 root_well_def()

```
template<typename P >
void smtrat::cadcells::operators::rules::root_well_def (
            datastructures::SampledDerivation< P > & deriv,
            datastructures::IndexedRoot root )
```

14.35 smtrat::cadcells::representation Namespace Reference

Namespaces

- util

Data Structures

- struct cell

  Note: If connected(i) holds, then the indexed root ordering must contain an ordering between the interval bounds.
- struct cell< CellHeuristic::BIGGEST_CELL >
- struct cell< CellHeuristic::CHAIN_EQ >
- struct cell< CellHeuristic::LOWEST_DEGREE_BARRIERS >
- struct cell< CellHeuristic::LOWEST_DEGREE_BARRIERS_EQ >
- struct covering
- struct covering< CoveringHeuristic::CHAIN_COVERING >
- struct covering< CoveringHeuristic::DEFAULT_COVERING >
- struct delineation
- struct delineation< DelineationHeuristic::CHAIN >

Enumerations

- enum CellHeuristic { BIGGEST_CELL, CHAIN_EQ, LOWEST_DEGREE_BARRIERS, LOWEST_DEGREE_BARRIERS_EQ }
- enum CoveringHeuristic { DEFAULT_COVERING, CHAIN_COVERING }
- enum DelineationHeuristic { CHAIN }

Functions

- std::string get_name (CellHeuristic heuristic)
- std::string get_name (CoveringHeuristic heuristic)
- std::string get_name (DelineationHeuristic heuristic)
- std::ostream & operator<< (std::ostream &os, CellHeuristic heuristic)
- std::ostream & operator<< (std::ostream &os, CoveringHeuristic heuristic)
- std::ostream & operator<< (std::ostream &os, DelineationHeuristic 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)
- template<typename T >
  void compute_barriers (datastructures::SampledDerivationRef< T > &der, datastructures::CellRepresentation< T > &response, bool section)
- 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)

14.35.1 Enumeration Type Documentation

14.35.1.1 CellHeuristic `enum smtrat::cadcells::representation::CellHeuristic`

Enumerator

BIGGEST_CELL	
CHAIN_EQ	
LOWEST_DEGREE_BARRIERS	
LOWEST_DEGREE_BARRIERS_EQ	

14.35.1.2 CoveringHeuristic `enum smtrat::cadcells::representation::CoveringHeuristic`

Enumerator

DEFAULT_COVERING	
CHAIN_COVERING	

14.35.1.3 DelineationHeuristic `enum smtrat::cadcells::representation::DelineationHeuristic`

Enumerator

CHAIN	

14.35.2 Function Documentation

14.35.2.1 compute_barriers() `template<typename T >`

```
void smtrat::cadcells::representation::compute_barriers (
```

```
datastructures::SampledDerivationRef< T > & der,
datastructures::CellRepresentation< T > & response,
bool section )  [inline]
```

14.35.2.2 compute_default_ordering()

```
template<typename T >
datastructures::IndexedRootOrdering smtrat::cadcells::representation::compute_default_ordering
(
    const std::vector< datastructures::CellRepresentation< T >> & cells )
```

14.35.2.3 compute_min_ders()

```
template<typename T >
std::vector<datastructures::SampledDerivationRef<T> > smtrat::cadcells::representation↩
::compute_min_ders (
    const std::vector< datastructures::SampledDerivationRef< T >> & ders )
```

14.35.2.4 compute_section_all_equational()

```
template<typename T >
void smtrat::cadcells::representation::compute_section_all_equational (
    datastructures::SampledDerivationRef< T > & der,
    datastructures::CellRepresentation< T > & response )  [inline]
```

14.35.2.5 get_name() [1/3]

```
std::string smtrat::cadcells::representation::get_name (
    CellHeuristic heuristic )  [inline]
```

14.35.2.6 get_name() [2/3]

```
std::string smtrat::cadcells::representation::get_name (
    CoveringHeuristic heuristic )  [inline]
```

14.35.2.7 get_name() [3/3]

```
std::string smtrat::cadcells::representation::get_name (
    DelineationHeuristic heuristic )  [inline]
```

14.35.2.8 maintain_connectedness()

```
template<typename T >
void smtrat::cadcells::representation::maintain_connectedness (
    datastructures::SampledDerivationRef< T > & der,
    datastructures::CellRepresentation< T > & response )
```

14.35.2.9 operator<<() [1/3]

```
std::ostream& smtrat::cadcells::representation::operator<< (
    std::ostream & os,
    CellHeuristic heuristic )  [inline]
```

14.35.2.10 operator<<() [2/3]

```
std::ostream& smtrat::cadcells::representation::operator<< (
    std::ostream & os,
    CoveringHeuristic heuristic )  [inline]
```

14.35.2.11 operator<<() [3/3]

```
std::ostream& smtrat::cadcells::representation::operator<< (
    std::ostream & os,
    DelineationHeuristic heuristic ) [inline]
```

14.36 smtrat::cadcells::representation::util Namespace Reference

Functions

- bool compare_simplest (datastructures::Projections &proj, datastructures::PolyRef p1, datastructures::PolyRef p2)
- std::optional< datastructures::IndexedRoot > simplest_bound (datastructures::Projections &proj, const std::vector< datastructures::IndexedRoot > &bounds, const boost::container::flat_set< datastructures::PolyRef > &ignoring)
- datastructures::IndexedRoot simplest_bound (datastructures::Projections &proj, const std::vector< datastructures::IndexedRoot > &bounds)
- datastructures::CellDescription compute_simplest_cell (datastructures::Projections &proj, const datastructures::DelineationInterv &del)
- std::optional< datastructures::IndexedRootOrdering > simplest_delineation_ordering (datastructures::Projections &proj, datastructures::Delineation &delin)

14.36.1 Function Documentation

14.36.1.1 compare_simplest()

```
bool smtrat::cadcells::representation::util::compare_simplest (
    datastructures::Projections & proj,
    datastructures::PolyRef p1,
    datastructures::PolyRef p2 ) [inline]
```

14.36.1.2 compute_simplest_cell()

```
datastructures::CellDescription smtrat::cadcells::representation↩
::util::compute_simplest_cell (
    datastructures::Projections & proj,
    const datastructures::DelineationInterval & del ) [inline]
```

14.36.1.3 simplest_bound() [1/2]

```
datastructures::IndexedRoot smtrat::cadcells::representation↩
::util::simplest_bound (
    datastructures::Projections & proj,
    const std::vector< datastructures::IndexedRoot > & bounds ) [inline]
```

14.36.1.4 simplest_bound() [2/2]

```
std::optional<datastructures::IndexedRoot> smtrat::cadcells↩
::representation::util::simplest_bound (
    datastructures::Projections & proj,
    const std::vector< datastructures::IndexedRoot > & bounds,
    const boost::container::flat_set< datastructures::PolyRef > & ignoring ) [inline]
```

14.36.1.5 simplest_delineation_ordering()

```
std::optional<datastructures::IndexedRootOrdering>
smtrat::cadcells::representation::util::simplest_delineation_ordering (
    datastructures::Projections & proj,
    datastructures::Delineation & delin ) [inline]
```

14.37 smtrat::compile_information Namespace Reference

Compile time generated information about compiler and system version.

Variables

- const std::string SystemName = "Linux"
- const std::string SystemVersion = "5.15.0-0.bpo.3-amd64"
- const std::string BuildType = "DEBUG"
- const std::string CXXCompiler = "/usr/bin/c++"
- const std::string CXXCompilerVersion = "9.3.0"
- const std::string GitRevisionSHA1 = "d18e87a5e120a78f89c110e0271ce92df17a2f36"
- const std::string PackageName = "smtrat"
- const std::string ProjectName = "SMT-RAT"
- const std::string Version = "22.04"
- const std::string Website = "https://github.com/smtrat/smtrat/wiki"
- const std::string GitVersion = "22.04"

14.37.1 Detailed Description

Compile time generated information about compiler and system version.

14.37.2 Variable Documentation

14.37.2.1 BuildType

`const std::string smtrat::compile_information::BuildType = "DEBUG"`

14.37.2.2 CXXCompiler

`const std::string smtrat::compile_information::CXXCompiler = "/usr/bin/c++"`

14.37.2.3 CXXCompilerVersion

`const std::string smtrat::compile_information::CXXCompilerVersion = "9.3.0"`

14.37.2.4 GitRevisionSHA1

`const std::string smtrat::compile_information::GitRevisionSHA1 = "d18e87a5e120a78f89c110e0271ce92df17a2f36"`

14.37.2.5 GitVersion

`const std::string smtrat::compile_information::GitVersion = "22.04"`

14.37.2.6 PackageName

`const std::string smtrat::compile_information::PackageName = "smtrat"`

14.37.2.7 ProjectName

`const std::string smtrat::compile_information::ProjectName = "SMT-RAT"`

14.37.2.8 SystemName

`const std::string smtrat::compile_information::SystemName = "Linux"`

14.37.2.9 SystemVersion

`const std::string smtrat::compile_information::SystemVersion = "5.15.0-0.bpo.3-amd64"`

14.37.2.10 Version

`const std::string smtrat::compile_information::Version = "22.04"`

14.37.2.11 Website `const std::string smtrat::compile_information::Website = "https://github.com/smtrat/smtrat/wiki"`

14.38 smtrat::datastructures Namespace Reference

Data Structures

- class BinaryHeap
- class DynHeap
- struct OrderPair
- struct OrderPair< T1, T2, true >

14.39 smtrat::execution Namespace Reference

Data Structures

- struct Assertion
- class ExecutionState
- struct Objective
- struct SoftAssertion

Enumerations

- enum Mode { START =0, ASSERT =1, SAT =2, UNSAT =3 }

14.39.1 Enumeration Type Documentation

14.39.1.1 Mode `enum smtrat::execution::Mode`

Enumerator

START	
ASSERT	
SAT	
UNSAT	

14.40 smtrat::expression Namespace Reference

Namespaces

- simplifier

Data Structures

- struct BinaryExpression
- class Expression
- struct ExpressionContent
- struct ExpressionConverter
- class ExpressionModifier
- class ExpressionPool
- struct ExpressionTypeChecker
- class ExpressionVisitor
- struct ITEExpression
- struct NaryExpression
- struct QuantifierExpression
- struct UnaryExpression

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)

14.40.1 Typedef Documentation

14.40.1.1 Expressions `typedef std::vector<Expression> smtrat::expression::Expressions`

14.40.2 Enumeration Type Documentation

14.40.2.1 BinaryType `enum smtrat::expression::BinaryType`

14.40.2.2 ITEType `enum smtrat::expression::ITEType`

Enumerator

ITE	

14.40.2.3 NaryType `enum smtrat::expression::NaryType`

Enumerator

AND	
OR	
XOR	
IFF	

14.40.2.4 QuantifierType `enum smtrat::expression::QuantifierType`

Enumerator

EXISTS	
FORALL	

14.40.2.5 UnaryType `enum smtrat::expression::UnaryType`

Enumerator

NOT	

14.40.3 Function Documentation

14.40.3.1 operator<<() [1/13]

```
std::ostream& smtrat::expression::operator<< (
    std::ostream & os,
    BinaryType type ) [inline]
```

14.40.3.2 operator<<() [2/13]

```
std::ostream& smtrat::expression::operator<< (
    std::ostream & os,
    const BinaryExpression & expr ) [inline]
```

14.40.3.3 operator<<() [3/13]

```
std::ostream & smtrat::expression::operator<< (
    std::ostream & os,
    const Expression & expr )
```

14.40.3.4 operator<<() [4/13]

```
std::ostream& smtrat::expression::operator<< (
    std::ostream & os,
    const ExpressionContent & expr ) [inline]
```

14.40.3.5 operator<<() [5/13]

```
std::ostream& smtrat::expression::operator<< (
    std::ostream & os,
    const ExpressionContent * expr ) [inline]
```

14.40.3.6 operator<<() [6/13]

```
std::ostream& smtrat::expression::operator<< (
    std::ostream & os,
    const ITEExpression & expr ) [inline]
```

14.40.3.7 operator<<() [7/13]

```
std::ostream& smtrat::expression::operator<< (
    std::ostream & os,
    const NaryExpression & expr ) [inline]
```

14.40.3.8 operator<<() [8/13]

```
std::ostream& smtrat::expression::operator<< (
    std::ostream & os,
    const QuantifierExpression & expr ) [inline]
```

14.40.3.9 operator<<() [9/13]

```
std::ostream& smtrat::expression::operator<< (
    std::ostream & os,
    const UnaryExpression & expr ) [inline]
```

14.40.3.10 operator<<() [10/13]

```
std::ostream& smtrat::expression::operator<< (
    std::ostream & os,
    ITEType ) [inline]
```

14.40.3.11 operator<<() [11/13]

```
std::ostream& smtrat::expression::operator<< (
    std::ostream & os,
    NaryType type ) [inline]
```

14.40.3.12 operator<<() [12/13]

```
std::ostream& smtrat::expression::operator<< (
    std::ostream & os,
    QuantifierType type ) [inline]
```

14.40.3.13 operator<<() [13/13]

```
std::ostream& smtrat::expression::operator<< (
    std::ostream & os,
    UnaryType type ) [inline]
```

14.41 smtrat::expression::simplifier Namespace Reference

Data Structures

- struct BaseSimplifier
- struct DuplicateSimplifier
- struct MergeSimplifier
- struct NegationSimplifier
- class Simplifier
- struct SimplifierChainCaller
- struct SimplifierChainCaller< 0 >
- struct SingletonSimplifier

Typedefs

- typedef std::tuple< MergeSimplifier, DuplicateSimplifier, SingletonSimplifier > SimplifierChain

14.41.1 Typedef Documentation

14.41.1.1 SimplifierChain

```
typedef std::tuple< MergeSimplifier, DuplicateSimplifier, SingletonSimplifier
> smtrat::expression::simplifier::SimplifierChain
```

14.42 smtrat::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)

14.42.1 Typedef Documentation

14.42.1.1 RewriteRules `typedef std::map<carl::Variable, std::pair<TermT, carl::BitVector> > smtrat::groebner::RewriteRules`

14.42.2 Function Documentation

14.42.2.1 rewritePolynomial() `template<typename Polynomial >`

```
static Polynomial smtrat::groebner::rewritePolynomial (
            const Polynomial & inputPolynomial,
            const RewriteRules & rules )  [static]
```

14.43 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 LRAModule< LRASettings1 >::LRAVariable LRAVariable
- typedef std::set< const IcpVariable *, icpVariableComp > set_icpVariable

Enumerations

- enum Updated { Updated::LEFT, Updated::RIGHT, Updated::BOTH, Updated::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 LRAModule< LRASettings1 >::LRAVariable * getOriginalLraVar (carl::Variable::Arg _var, const LRAModule< LRASettings1 > &_lra)

14.43.1 Typedef Documentation

14.43.1.1 Contractor `template<template< typename > class Operator>`

```
using smtrat::icp::Contractor = typedef carl::Contraction<Operator, Poly>
```

14.43.1.2 LRAVariable `typedef LRAModule<LRASettings1>::LRAVariable smtrat::icp::LRAVariable`

14.43.1.3 set_icpVariable `typedef std::set<const IcpVariable*, icpVariableComp> smtrat::icp::set_icpVariable`

14.43.2 Enumeration Type Documentation

14.43.2.1 Updated `enum smtrat::icp::Updated [strong]`

Enumerator

LEFT	
RIGHT	
BOTH	
NONE	

14.43.3 Function Documentation

14.43.3.1 getNonlinearMonomials()

```
std::vector< Poly > smtrat::icp::getNonlinearMonomials (
    const Poly & _expr )
```

Obtains the non-linear monomials of the given polynomial.

Parameters

_expr	The polynomial to obtain the non-linear monomials for.

Returns The non-linear monomials.

14.43.3.2 getOriginalLraVar()

```
const LRAModule< LRASettings1 >::LRAVariable * smtrat::icp::get↩
OriginalLraVar (
    carl::Variable::Arg _var,
    const LRAModule< LRASettings1 > & _lra )
```

14.43.3.3 intervalBoxContainsEmptyInterval()

```
bool smtrat::icp::intervalBoxContainsEmptyInterval
(
    const EvalDoubleIntervalMap & _intervals )
```

Checks mIntervals if it contains an empty interval.

Returns

14.43.3.4 intervalToConstraint()

```
std::pair< ConstraintT, ConstraintT > smtrat::icp::intervalTo↩
Constraint (
    const Poly & _lhs,
    const smtrat::DoubleInterval _interval )
```

Creates a new constraint from an existing interval.

Parameters

_interval	

Returns pair <lowerBoundConstraint*, upperBoundConstraint*>

14.44 smtrat::impl Namespace Reference

Data Structures

- class fixedsize_allocator_freelists
- class fixedsize_allocator_impl
- class fixedsize_allocator_impl< T, SizeShift, true, true >
- class fixedsize_allocator_size_helper
- class fixedsize_freelists

  Publicly derive from all fixedsize_freelist types with chunk sizes between SizeCurrent and SizeMax.
- class fixedsize_freelists< SizeCurrent, SizeMax, true >

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

14.44.1 Function Documentation

14.44.1.1 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).

14.44.2 Variable Documentation

14.44.2.1 LOWER_SIZE_BOUND

```
CONSTEXPR std::size_t smtrat::impl::LOWER_SIZE_BOUND = sizeof(void*)
[static]
```

14.44.2.2 UPPER_SIZE_BOUND

```
CONSTEXPR std::size_t smtrat::impl::UPPER_SIZE_BOUND = 256 [static]
```

14.45 smtrat::lra Namespace Reference

Data Structures

- class Bound

  Stores a bound, which could be an upper "<= b" or a lower bound ">= b" for a bound value b.
- class Numeric
- class Tableau
- class TableauEntry
- class Value
- class Variable

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)

  Calculates the least common multiple of the two arguments.
- Numeric gcd (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)

  Multiplies 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)

  Decrements 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)

Variables

- static EntryID LAST_ENTRY_ID = 0

14.45.1 Typedef Documentation

14.45.1.1 EntryID `typedef size_t smtrat::lra::EntryID`

14.45.2 Function Documentation

14.45.2.1 abs()

```
Numeric smtrat::lra::abs (
    const Numeric & _value )
```

Calculates the absolute value of the given Numeric.

Parameters

_value	The Numeric to calculate the Numeric for.

Returns The absolute value of the given Numeric.

14.45.2.2 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

_valueA	An integer.
_valueB	An integer.

Returns The least common divisor of the two arguments.

14.45.2.3 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

_valueA	An integer.
_valueB	An integer.

Returns The least common multiple of the two arguments.

14.45.2.4 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

_valueA	An integer.
_valueB	An integer != 0.

Returns The first argument modulo the second argument.

14.45.2.5 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.

14.45.2.6 operator*=()

```
Numeric & smtrat::lra::operator*= (
    Numeric & _valueA,
    const Numeric & _valueB )
```

Multiplies the second given Numeric to the first given Numeric.

Parameters

_valueA	The Numeric to multiply.
_valueB	The Numeric to multiply by.

Returns The first given Numeric multiplied by the second given Numeric.

14.45.2.7 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.

14.45.2.8 operator++()

```
Numeric & smtrat::lra::operator++ (
    Numeric & _value )
```

Increments the given Numeric by one.

Parameters

_value	The Numeric to increment.

Returns The given Numeric incremented by one.

14.45.2.9 operator+=()

```
Numeric & smtrat::lra::operator+= (
    Numeric & _valueA,
    const Numeric & _valueB )
```

Adds the value of the second given Numeric to the second given Numeric.

Parameters

_valueA	The Numeric to add to.
_valueB	The Numeric to add.

Returns The first given Numeric increased by the second given Numeric.

14.45.2.10 operator-() [1/2]

```
Numeric smtrat::lra::operator- (
    const Numeric & _value )
```

Calculates the additive inverse of the given Numeric.

Parameters

_value	The Numeric to calculate the additive inverse for.

Returns The additive inverse of the given Numeric.

14.45.2.11 operator-() [2/2]

```
Numeric smtrat::lra::operator- (
    const Numeric & _valueA,
    const Numeric & _valueB )
```

Calculates the difference between the two given Numerics.

Parameters

_valueA	The minuend.
_valueB	The subtrahend.

Returns The difference between the two given Numerics.

14.45.2.12 operator--()

```
Numeric & smtrat::lra::operator-- (
    Numeric & _value )
```

Decrements the given Numeric by one.

Parameters

_value	The Numeric to decrement.

Returns The given Numeric decremented by one.

14.45.2.13 operator-=()

```
Numeric & smtrat::lra::operator-= (
    Numeric & _valueA,
    const Numeric & _valueB )
```

Subtracts the second given Numeric to the first given Numeric.

Parameters

_valueA	The Numeric to subtract from.
_valueB	The Numeric to subtract.

Returns The first given Numeric subtracted by the second given Numeric.

14.45.2.14 operator/()

```
Numeric smtrat::lra::operator/ (
    const Numeric & _valueA,
    const Numeric & _valueB )
```

Calculates the division of the two given Numerics.

Parameters

_valueA	The dividend.
_valueB	The divisor.

Returns The difference of the two given Numerics.

14.45.2.15 operator/=()

```
Numeric & smtrat::lra::operator/= (
    Numeric & _valueA,
    const Numeric & _valueB )
```

Divides the first given Numeric by the second given Numeric.

Parameters

_valueA	The Numeric to divide.
_valueB	The Numeric to divide by.

Returns The first given Numeric divided by the second given Numeric.

14.45.2.16 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

_out	The output stream to print on.
_value	The Numeric to print.

Returns The output stream after printing the given Numerics representation on it.

14.45.2.17 operator<<() [2/2]

```
template<typename T1 >
std::ostream& smtrat::lra::operator<< (
    std::ostream & _out,
    const Value< T1 > & _value )
```

14.45.3 Variable Documentation

14.45.3.1 LAST_ENTRY_ID

```
EntryID smtrat::lra::LAST_ENTRY_ID = 0 [static]
```

14.46 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 carl::RealAlgebraicNumber< Rational > &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)

14.46.1 Function Documentation

14.46.1.1 evaluate()

```
Rational smtrat::lve::evaluate (
    carl::Variable v,
    const Poly & p,
    const Rational & r )
```

14.46.1.2 get_non_root()

```
ModelValue smtrat::lve::get_non_root (
    carl::Variable v,
    const Poly & p )
```

14.46.1.3 get_root()

```
std::optional<ModelValue> smtrat::lve::get_root (
    carl::Variable v,
    const Poly & p )
```

14.46.1.4 get_value_for_sgn()

```
std::optional<ModelValue> smtrat::lve::get_value_for_sgn (
    carl::Variable v,
    const Poly & p,
    carl::Sign sign )
```

14.46.1.5 sgn()

```
carl::Sign smtrat::lve::sgn (
    carl::Variable v,
    const Poly & p,
    const carl::RealAlgebraicNumber< Rational > & r )
```

14.46.1.6 sgn_of_invariant_poly()

```
carl::Sign smtrat::lve::sgn_of_invariant_poly (
    carl::Variable v,
    const Poly & p )
```

14.47 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 >

14.47.1 Detailed Description

Contains strategy implementations for max SMT computations.

14.48 smtrat::mcsat Namespace Reference

Namespaces

- arithmetic
- constraint_type
- fm

  A Fourier-Motzkin based backend.
- icp
- nlsat
- onecell
- onecellcad
- smtaf
- variableordering
- vs

Data Structures

- 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:

- 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 InformationGetter
- class MCSATBackend
- class MCSATMixin
- struct ParallelExplanation
- struct SequentialAssignment
- struct SequentialExplanation
- struct TheoryLevel
- class VariableSelector

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, ModelValue > >
- using AssignmentOrConflict = std::variant< ModelValues, FormulasT >
- using Explanation = std::variant< FormulaT, ClauseChain >

Enumerations

- enum ConstraintType { ConstraintType::Constant, ConstraintType::Assigned, ConstraintType::Univariate, ConstraintType::Unassigned }

  This type categorizes constraints with respect to a given (partial) model and the next variable to be assigned.

- enum VariableOrdering { VariableOrdering::GreedyMaxUnivariate, VariableOrdering::FeatureBased, VariableOrdering::FeatureBasedZ3, VariableOrdering::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)
- std::ostream & operator<< (std::ostream &os, const VariableSelector &vs)

14.48.1 Typedef Documentation

14.48.1.1 AssignmentOrConflict

```
using smtrat::mcsat::AssignmentOrConflict = typedef std::variant<ModelValues,F
```

14.48.1.2 Explanation

`using smtrat::mcsat::Explanation = typedef std::variant<FormulaT, ClauseChain>`

14.48.1.3 ModelValues

`using smtrat::mcsat::ModelValues = typedef std::vector<std::pair<carl↩ ::Variable, ModelValue> >`

14.48.1.4 SharedVector

`typedef vector<std::optional<Explanation> , ShmemAllocator> smtrat::mcsat::SharedVect`

14.48.1.5 ShmemAllocator

`typedef allocator<std::optional<Explanation> , managed_shared_↩ memory::segment_manager> smtrat::mcsat::ShmemAllocator`

14.48.2 Enumeration Type Documentation

14.48.2.1 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.

14.48.2.2 VariableOrdering

`enum smtrat::mcsat::VariableOrdering [strong]`

Enumerator

GreedyMaxUnivariate	
FeatureBased	
FeatureBasedZ3	
FeatureBasedBrown	

14.48.3 Function Documentation

14.48.3.1 _transformToImplicationChain()

```
FormulaT smtrat::mcsat::_transformToImplicationChain (
    const FormulaT & formula,
    const Model & model,
    ClauseChain & chain,
    bool withEquivalences )
```

14.48.3.2 calculate_variable_order() [1/2]

```
template<VariableOrdering vot, typename Constraints >
std::vector<carl::Variable> smtrat::mcsat::calculate_variable_order (
            const Constraints & c )
```

14.48.3.3 calculate_variable_order() [2/2]

```
template<VariableOrdering vot>
std::vector<carl::Variable> smtrat::mcsat::calculate_variable_order (
            const std::vector< ConstraintT > & constraints )
```

14.48.3.4 get_name()

```
std::string smtrat::mcsat::get_name (
            VariableOrdering ordering ) [inline]
```

14.48.3.5 operator<<() [1/7]

```
std::ostream& smtrat::mcsat::operator<< (
            std::ostream & os,
            const Bookkeeping & bk ) [inline]
```

14.48.3.6 operator<<() [2/7]

```
template<typename Settings >
std::ostream& smtrat::mcsat::operator<< (
            std::ostream & os,
            const MCSATBackend< Settings > & backend )
```

14.48.3.7 operator<<() [3/7]

```
std::ostream& smtrat::mcsat::operator<< (
            std::ostream & os,
            const VariableSelector & vs ) [inline]
```

14.48.3.8 operator<<() [4/7]

```
std::ostream& smtrat::mcsat::operator<< (
            std::ostream & os,
            ConstraintType ct ) [inline]
```

14.48.3.9 operator<<() [5/7]

```
std::ostream& smtrat::mcsat::operator<< (
            std::ostream & os,
            VariableOrdering ordering ) [inline]
```

14.48.3.10 operator<<() [6/7]

```
std::ostream& smtrat::mcsat::operator<< (
            std::ostream & stream,
            const ClauseChain & chain ) [inline]
```

14.48.3.11 operator<<() [7/7]

```
std::ostream& smtrat::mcsat::operator<< (
            std::ostream & stream,
            const ClauseChain::Link & link ) [inline]
```

14.48.3.12 resolveExplanation()

```
FormulaT smtrat::mcsat::resolveExplanation (
    const Explanation & expl ) [inline]
```

14.49 smtrat::mcsat::arithmetic Namespace Reference

Data Structures

- struct AssignmentFinder
- class AssignmentFinder_detail
- 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)
- std::ostream & operator<< (std::ostream &os, const RootIndexer &ri)

14.49.1 Function Documentation

14.49.1.1 operator<<() [1/2]

```
std::ostream& smtrat::mcsat::arithmetic::operator<< (
    std::ostream & os,
    const Covering & ri ) [inline]
```

14.49.1.2 operator<<() [2/2]

```
std::ostream& smtrat::mcsat::arithmetic::operator<< (
    std::ostream & os,
    const RootIndexer & ri ) [inline]
```

14.50 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 isConstant (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 isUnivariate (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.

14.50.1 Function Documentation

14.50.1.1 categorize()

```
template<typename T >
ConstraintType smtrat::mcsat::constraint_type::categorize (
    const T & t,
    const Model & model,
    carl::Variable next )
```

14.50.1.2 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`.

14.50.1.3 isConstant() template<typename T >

```
bool smtrat::mcsat::constraint_type::isConstant (
            const T & t )
```

Checks whether the constraint is constant, i.e.

whether it contains no variables.

14.50.1.4 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.

14.50.1.5 isUnivariate() template<typename T >

```
bool smtrat::mcsat::constraint_type::isUnivariate (
            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.

14.51 smtrat::mcsat::fm Namespace Reference

A Fourier-Motzkin based backend.

Data Structures

- struct Bound
- struct ConflictGenerator
- struct DefaultComparator

  Does not order anything.
- struct DefaultSettings
- struct Explanation
- struct IgnoreCoreSettings
- 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.

Functions

- bool isSubset (const carl::Variables &subset, const carl::Variables &superset)
- std::ostream & operator<< (std::ostream &os, const Bound &b)

14.51.1 Detailed Description

A Fourier-Motzkin based backend.

Preprocessing of constraints:

The input is a constraint c: p∗x∼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∼q). If this is the case, we can return (c && p=0) -> 0∼q as explanation. Otherwise, we evaluate c over the partial model and obtain x∼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 >= 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 != can be in conflict with

- a bound e for a constraint with = iff i.r == e.r, then we return i.c && e.c -> i.q ∗ e.p != e.q ∗ i.p
- two bounds l, u with >= resp. <= 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) -> l.q ∗ i.p != i.q ∗ l.p

For all bounds b involved, we add b.p != 0 as side condition to the explanation.

14.51.2 Function Documentation

14.51.2.1 isSubset()

```
bool smtrat::mcsat::fm::isSubset (
    const carl::Variables & subset,
    const carl::Variables & superset ) [inline]
```

14.51.2.2 operator<<()

```
std::ostream& smtrat::mcsat::fm::operator<< (
    std::ostream & os,
    const Bound & b ) [inline]
```

14.52 smtrat::mcsat::icp Namespace Reference

Data Structures

- class Dependencies
- struct Explanation
- class IntervalPropagation
- struct QueueEntry

14.53 smtrat::mcsat::nlsat Namespace Reference

Namespaces

- helper

Data Structures

- struct Explanation
- class ExplanationGenerator

14.54 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 formualas into the easier to use objects of the Constraint class.

14.54.1 Function Documentation

14.54.1.1 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(..)”

14.54.1.2 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(..)”

14.54.1.3 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(..)”

14.54.1.4 buildFormulaFromVC()

```
FormulaT smtrat::mcsat::nlsat::helper::buildFormulaFromVC (
            VariableComparisonT && vc ) [inline]
```

Construct a formula representing a variable comparison.

Simplify to a regular constraint if possible.

14.54.1.5 convertToConstraints()

```
std::set<ConstraintT> smtrat::mcsat::nlsat::helper::convert↩
ToConstraints (
            std::vector< FormulaT > constraintAtoms ) [inline]
```

Transform constraints represented as atomic formualas into the easier to use objects of the Constraint class.

14.55 smtrat::mcsat::onecell Namespace Reference

Data Structures

- struct Explanation

14.56 smtrat::mcsat::onecellcad Namespace Reference

Namespaces

- firstlevel
- levelwise
- recursive

OneCell Explanation.

Data Structures

- class OneCellCAD
- 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.
- struct TagPoly

  Tagged Polynomials.

Typedefs

- using RAN = carl::RealAlgebraicNumber< smtrat::Rational >
- using RANMap = std::map< carl::Variable, RAN >
- using CADCell = std::vector< std::variant< Sector, Section > >

  Represent a single cell [brown15].

Enumerations

- enum InvarianceType { InvarianceType::ORD_INV, InvarianceType::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)
- 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)
- carl::RealAlgebraicPoint< smtrat::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)

14.56.1 Typedef Documentation

14.56.1.1 CADCell

`using smtrat::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.

14.56.1.2 RAN

`using smtrat::mcsat::onecellcad::RAN = typedef carl::RealAlgebraicNumber<smtrat::Rational>`

14.56.1.3 RANMap

`using smtrat::mcsat::onecellcad::RANMap = typedef std::map<carl::Variable, RAN>`

14.56.2 Enumeration Type Documentation

14.56.2.1 InvarianceType

`enum smtrat::mcsat::onecellcad::InvarianceType [strong]`

Invariance Types.

Enumerator

ORD_INV	
SIGN_INV	

14.56.3 Function Documentation

14.56.3.1 addResultants()

```
void smtrat::mcsat::onecellcad::addResultants (
    std::vector< std::pair< Poly, Poly >> & resultants,
    std::vector< std::vector< TagPoly >> & polys,
    carl::Variable mainVar,
    const std::vector< carl::Variable > & variableOrder ) [inline]
```

14.56.3.2 appendOnCorrectLevel()

```
void smtrat::mcsat::onecellcad::appendOnCorrectLevel (
    const Poly & poly,
    InvarianceType tag,
    std::vector< std::vector< TagPoly >> & polys,
    std::vector< carl::Variable > variableOrder ) [inline]
```

14.56.3.3 asMultiPolys()

```
std::vector<Poly> smtrat::mcsat::onecellcad::asMultiPolys (
    const std::vector< TagPoly > polys ) [inline]
```

14.56.3.4 asRANPoint()

```
carl::RealAlgebraicPoint<smtrat::Rational> smtrat::mcsat::onecellcad↩
::asRANPoint (
    const mcsat::Bookkeeping & data ) [inline]
```

14.56.3.5 asRootExpr()

```
MultivariateRootT smtrat::mcsat::onecellcad::asRootExpr (
    carl::Variable rootVariable,
    Poly poly,
    std::size_t rootIdx ) [inline]
```

Parameters

rootVariable	The variable with respect to which the roots are computed in the end. It will be replaced by the special unique root-variable "_z" common in root-expressions.

14.56.3.6 asVector()

```
template<typename T >
std::vector<T> smtrat::mcsat::onecellcad::asVector (
    const std::set< T > s )
```

14.56.3.7 cellDimension()

```
std::size_t smtrat::mcsat::onecellcad::cellDimension (
    const CADCell & cell,
    const std::size_t uptoLevel ) [inline]
```

Returns Dimension of a hybercube 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').

14.56.3.8 contains() [1/2]

```
template<typename T >
bool smtrat::mcsat::onecellcad::contains (
            const std::vector< T > & list,
            const T & elem )
```

14.56.3.9 contains() [2/2]

```
bool smtrat::mcsat::onecellcad::contains (
            const std::vector< TagPoly > & polys,
            const Poly & poly ) [inline]
```

14.56.3.10 discriminant()

```
Poly smtrat::mcsat::onecellcad::discriminant (
            const carl::Variable & mainVariable,
            const Poly & p ) [inline]
```

Projection related utilities for onecellcad.

14.56.3.11 duplicateElimination() [1/2]

```
void smtrat::mcsat::onecellcad::duplicateElimination (
            std::vector< Poly > & vec ) [inline]
```

14.56.3.12 duplicateElimination() [2/2]

```
std::vector<std::pair<Poly, Poly> > smtrat::mcsat↩
::onecellcad::duplicateElimination (
            std::vector< std::pair< Poly, Poly >> vec ) [inline]
```

14.56.3.13 fullSpaceCell()

```
CADCell smtrat::mcsat::onecellcad::fullSpaceCell (
            std::size_t cellComponentCount ) [inline]
```

14.56.3.14 getDegree()

```
std::size_t smtrat::mcsat::onecellcad::getDegree (
            TagPoly p,
            carl::Variable v ) [inline]
```

Parameters

p	Polynomial to get degree from
v	Rootvariable for degree calc

Returns

14.56.3.15 hasOnlyNonConstIrreducibles()

```
template<typename PolyType >
bool smtrat::mcsat::onecellcad::hasOnlyNonConstIrreducibles (
            const std::vector< PolyType > & polys )
```

14.56.3.16 hasUniqElems()

```
template<typename T >
bool smtrat::mcsat::onecellcad::hasUniqElems (
            const std::vector< T > & container )
```

14.56.3.17 isNonConstIrreducible()

```
template<typename PolyType >
bool smtrat::mcsat::onecellcad::isNonConstIrreducible (
            const PolyType & poly )
```

14.56.3.18 isSubset()

```
template<typename T >
bool smtrat::mcsat::onecellcad::isSubset (
            const std::vector< T > & c1,
            const std::vector< T > & c2 )
```

14.56.3.19 leadcoefficient()

```
Poly smtrat::mcsat::onecellcad::leadcoefficient (
            const carl::Variable & mainVariable,
            const Poly & p ) [inline]
```

14.56.3.20 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:
- 'poly.gatherVariables()' must be a subset of 'variableOrder'.

14.56.3.21 nonConstIrreducibleFactors()

```
std::vector<TagPoly> smtrat::mcsat::onecellcad::non↩
ConstIrreducibleFactors (
            std::vector< carl::Variable > variableOrder,
            Poly poly,
            InvarianceType tag ) [inline]
```

14.56.3.22 operator<()

```
bool smtrat::mcsat::onecellcad::operator< (
            const InvarianceType & l,
            const InvarianceType & r ) [inline]
```

14.56.3.23 operator<<() [1/7]

```
std::ostream& smtrat::mcsat::onecellcad::operator<< (
    std::ostream & os,
    const CADCell & cell )  [inline]
```

14.56.3.24 operator<<() [2/7]

```
std::ostream& smtrat::mcsat::onecellcad::operator<< (
    std::ostream & os,
    const InvarianceType & inv )  [inline]
```

14.56.3.25 operator<<() [3/7]

```
std::ostream& smtrat::mcsat::onecellcad::operator<< (
    std::ostream & os,
    const Section & s )  [inline]
```

14.56.3.26 operator<<() [4/7]

```
std::ostream& smtrat::mcsat::onecellcad::operator<< (
    std::ostream & os,
    const Sector & s )  [inline]
```

14.56.3.27 operator<<() [5/7]

```
std::ostream& smtrat::mcsat::onecellcad::operator<< (
    std::ostream & os,
    const std::vector< std::vector< TagPoly >> & lvls )  [inline]
```

14.56.3.28 operator<<() [6/7]

```
std::ostream& smtrat::mcsat::onecellcad::operator<< (
    std::ostream & os,
    const std::vector< TagPoly > & polys )  [inline]
```

14.56.3.29 operator<<() [7/7]

```
std::ostream& smtrat::mcsat::onecellcad::operator<< (
    std::ostream & os,
    const TagPoly & p )  [inline]
```

14.56.3.30 operator==()

```
bool smtrat::mcsat::onecellcad::operator== (
    const TagPoly & lhs,
    const TagPoly & rhs )  [inline]
```

14.56.3.31 optimized_singleLevelFullProjection()

```
bool smtrat::mcsat::onecellcad::optimized_↩
singleLevelFullProjection (
    carl::Variable mainVar,
    size_t currentLevel,
    std::vector< std::vector< TagPoly >> & projectionLevels,
    OneCellCAD & cad )  [inline]
```

14.56.3.32 polyVarsAreAllInList()

```
template<typename PolyType >
bool smtrat::mcsat::onecellcad::polyVarsAreAllInList (
            const std::vector< PolyType > & polys,
            const std::vector< carl::Variable > & variables )
```

14.56.3.33 prefix()

```
template<typename T >
std::vector<T> smtrat::mcsat::onecellcad::prefix (
            const std::vector< T > vars,
            std::size_t prefixSize )
```

14.56.3.34 resultant()

```
Poly smtrat::mcsat::onecellcad::resultant (
            const carl::Variable & mainVariable,
            const Poly & p1,
            const Poly & p2 )  [inline]
```

14.56.3.35 singleLevelFullProjection()

```
void smtrat::mcsat::onecellcad::singleLevelFullProjection
(
            std::vector< carl::Variable > & variableOrder,
            carl::Variable mainVar,
            size_t currentLevel,
            std::vector< std::vector< TagPoly >> & projectionLevels )  [inline]
```

14.57 smtrat::mcsat::onecellcad::firstlevel Namespace Reference

Data Structures

- struct covering_at_level
- struct indexed_root
- class root_indexer
- struct section
- struct sector

Typedefs

- using RAN = carl::RealAlgebraicNumber< Rational >
- using cell_at_level = std::variant< section, sector >
- using projection_results = std::map< Poly, InvarianceType >

Functions

- std::ostream & operator<< (std::ostream &os, const indexed_root &data)
- std::ostream & operator<< (std::ostream &os, const section &data)
- std::ostream & operator<< (std::ostream &os, const sector &data)
- std::ostream & operator<< (std::ostream &os, const cell_at_level &data)
- std::ostream & operator<< (std::ostream &os, const cell_at_level *data)
- std::ostream & operator<< (std::ostream &os, const root_indexer::sample &data)
- std::ostream & operator<< (std::ostream &os, const root_indexer &data)
- auto as_ran_map (const Model &model)
- bool var_subset (const carl::carlVariables &vars, const Model &model, carl::Variable var)
- bool compute_unsat_intervals (const VariableComparisonT &constr, const Model &model, carl::Variable variable, std::vector< cell_at_level > &results)
- bool compute_unsat_intervals (const ConstraintT &constr, const Model &model, carl::Variable variable, std↩::vector< cell_at_level > &results)
- bool compute_unsat_intervals (const FormulaT &constr, const Model &model, carl::Variable variable, std↩::vector< cell_at_level > &results)
- void insert_projection_result (const Poly &respoly, InvarianceType type, projection_results &results)

- void required_coefficients (const Poly &mpoly, const Model &model, carl::Variable variable, projection_results &results)
- void project (const sector &cell, const Model &model, carl::Variable variable, projection_results &results)
- void project (const section &cell, const Model &model, carl::Variable variable, projection_results &results)
- void project (const cell_at_level &cell, const Model &model, carl::Variable variable, projection_results &results)
- std::ostream & operator<< (std::ostream &os, const covering_at_level &data)
- void project (const covering_at_level &covering, const Model &model, carl::Variable variable, projection_results &results)
- covering_at_level compute_covering (const std::vector< cell_at_level > &cells)
- std::optional< std::vector< TagPoly > > first_level_projection (const FormulasT &input, const Model &model, carl::Variable variable)

14.57.1 Typedef Documentation

14.57.1.1 cell_at_level

```
using smtrat::mcsat::onecellcad::firstlevel::cell_at_level = typedef std↩
::variant<section,sector>
```

14.57.1.2 projection_results

```
using smtrat::mcsat::onecellcad::firstlevel::projection_results =
typedef std::map<Poly, InvarianceType>
```

14.57.1.3 RAN

```
using smtrat::mcsat::onecellcad::firstlevel::RAN = typedef carl::RealAlgebraic↩
Number<Rational>
```

14.57.2 Function Documentation

14.57.2.1 as_ran_map()

```
auto smtrat::mcsat::onecellcad::firstlevel::as_ran_map (
            const Model & model )
```

14.57.2.2 compute_covering()

```
covering_at_level smtrat::mcsat::onecellcad::firstlevel::compute_↩
covering (
            const std::vector< cell_at_level > & cells )
```

14.57.2.3 compute_unsat_intervals() [1/3]

```
bool smtrat::mcsat::onecellcad::firstlevel::compute_↩
unsat_intervals (
            const ConstraintT & constr,
            const Model & model,
            carl::Variable variable,
            std::vector< cell_at_level > & results )
```

14.57.2.4 compute_unsat_intervals() [2/3]

```
bool smtrat::mcsat::onecellcad::firstlevel::compute_↩
unsat_intervals (
            const FormulaT & constr,
            const Model & model,
            carl::Variable variable,
            std::vector< cell_at_level > & results )
```

14.57.2.5 compute_unsat_intervals() [3/3]

```
bool smtrat::mcsat::onecellcad::firstlevel::compute_unsat_intervals (
            const VariableComparisonT & constr,
            const Model & model,
            carl::Variable variable,
            std::vector< cell_at_level > & results )
```

14.57.2.6 first_level_projection()

```
std::optional<std::vector<TagPoly> > smtrat::mcsat::onecellcad::firstlevel::first_level_projection (
            const FormulasT & input,
            const Model & model,
            carl::Variable variable )
```

14.57.2.7 insert_projection_result()

```
void smtrat::mcsat::onecellcad::firstlevel::insert_projection_result (
            const Poly & respoly,
            InvarianceType type,
            projection_results & results )
```

14.57.2.8 operator<<() [1/8]

```
std::ostream& smtrat::mcsat::onecellcad::firstlevel::operator<< (
            std::ostream & os,
            const cell_at_level & data )
```

14.57.2.9 operator<<() [2/8]

```
std::ostream& smtrat::mcsat::onecellcad::firstlevel::operator<< (
            std::ostream & os,
            const cell_at_level * data )
```

14.57.2.10 operator<<() [3/8]

```
std::ostream& smtrat::mcsat::onecellcad::firstlevel::operator<< (
            std::ostream & os,
            const covering_at_level & data )
```

14.57.2.11 operator<<() [4/8]

```
std::ostream& smtrat::mcsat::onecellcad::firstlevel::operator<< (
            std::ostream & os,
            const indexed_root & data )
```

14.57.2.12 operator<<() [5/8]

```
std::ostream& smtrat::mcsat::onecellcad::firstlevel::operator<< (
            std::ostream & os,
            const root_indexer & data )
```

14.57.2.13 operator<<() [6/8]

```
std::ostream& smtrat::mcsat::onecellcad::firstlevel::operator<<
(
    std::ostream & os,
    const root_indexer::sample & data )
```

14.57.2.14 operator<<() [7/8]

```
std::ostream& smtrat::mcsat::onecellcad::firstlevel::operator<<
(
    std::ostream & os,
    const section & data )
```

14.57.2.15 operator<<() [8/8]

```
std::ostream& smtrat::mcsat::onecellcad::firstlevel::operator<<
(
    std::ostream & os,
    const sector & data )
```

14.57.2.16 project() [1/4]

```
void smtrat::mcsat::onecellcad::firstlevel::project (
    const cell_at_level & cell,
    const Model & model,
    carl::Variable variable,
    projection_results & results )
```

14.57.2.17 project() [2/4]

```
void smtrat::mcsat::onecellcad::firstlevel::project (
    const covering_at_level & covering,
    const Model & model,
    carl::Variable variable,
    projection_results & results )
```

14.57.2.18 project() [3/4]

```
void smtrat::mcsat::onecellcad::firstlevel::project (
    const section & cell,
    const Model & model,
    carl::Variable variable,
    projection_results & results )
```

14.57.2.19 project() [4/4]

```
void smtrat::mcsat::onecellcad::firstlevel::project (
    const sector & cell,
    const Model & model,
    carl::Variable variable,
    projection_results & results )
```

14.57.2.20 required_coefficients()

```
void smtrat::mcsat::onecellcad::firstlevel::required_coefficients
(
    const Poly & mpoly,
    const Model & model,
    carl::Variable variable,
    projection_results & results )
```

14.57.2.21 var_subset()

```
bool smtrat::mcsat::onecellcad::firstlevel::var_subset (
    const carl::carlVariables & vars,
    const Model & model,
    carl::Variable var )
```

14.58 smtrat::mcsat::onecellcad::levelwise Namespace Reference

Data Structures

- struct Explanation
- class LevelwiseCAD
- struct SectionHeuristic1
- struct SectionHeuristic2
- struct SectionHeuristic3
- struct SectorHeuristic1
- struct SectorHeuristic2
- struct SectorHeuristic3

14.59 smtrat::mcsat::onecellcad::recursive Namespace Reference

OneCell Explanation.

Data Structures

- struct CoverNullification
- struct DegreeAscending
- struct DegreeDescending
- struct DontCoverNullification
- struct Explanation
- struct NoHeuristic
- class RecursiveCAD

Enumerations

- enum ShrinkResult { ShrinkResult::SUCCESS, ShrinkResult::FAIL }

Functions

- void setNullification (bool n)
- std::ostream & operator<< (std::ostream &os, const ShrinkResult &s)

Variables

- bool cover_nullification

14.59.1 Detailed Description

OneCell Explanation.

14.59.1.1 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.

14.59.1.2 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.

14.59.1.3 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.

14.59.1.4 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.

14.59.1.5 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 .

14.59.1.6 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.

14.59.1.7 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.

14.59.1.8 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.

14.59.2 Enumeration Type Documentation

14.59.2.1 ShrinkResult `enum smtrat::mcsat::onecellcad::recursive::ShrinkResult [strong]`

Enumerator

SUCCESS	
FAIL	

14.59.3 Function Documentation

14.59.3.1 operator<<()
```
std::ostream& smtrat::mcsat::onecellcad::recursive::operator<< (
    std::ostream & os,
    const ShrinkResult & s ) [inline]
```

14.59.3.2 setNullification()
```
void smtrat::mcsat::onecellcad::recursive::setNullification (
    bool n ) [inline]
```

14.59.4 Variable Documentation

14.59.4.1 cover_nullification `bool smtrat::mcsat::onecellcad::recursive::cover_nullification`

14.60 smtrat::mcsat::smtaf Namespace Reference

Data Structures

- struct AssignmentFinder
- class AssignmentFinder_SMT
- struct DefaultSettings

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)

14.60.1 Typedef Documentation

14.60.1.1 VariablePos

```
using smtrat::mcsat::smtaf::VariablePos = typedef std::vector<carl::↩
Variable>::const_iterator
```

14.60.1.2 VariableRange

```
using smtrat::mcsat::smtaf::VariableRange = typedef std::pair<VariablePos,VariablePos
```

14.60.2 Function Documentation

14.60.2.1 includes()

```
bool smtrat::mcsat::smtaf::includes (
    const VariableRange & superset,
    const carl::Variables & subset ) [inline]
```

14.61 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)

14.61.1 Function Documentation

14.61.1.1 feature_based()

```
template<typename Constraints >
std::vector<carl::Variable> smtrat::mcsat::variableordering::feature_based (
    const Constraints & c )
```

14.61.1.2 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.

14.61.1.3 feature_based_z3()

```
template<typename Constraints >
std::vector<carl::Variable> smtrat::mcsat::variableordering::feature_based_z3 (
            const Constraints & c )
```

14.61.1.4 gatherVariables()

```
template<typename Constraints >
void smtrat::mcsat::variableordering::gatherVariables (
            carl::carlVariables & vars,
            const Constraints & constraints )
```

14.61.1.5 greedy_max_univariate()

```
template<typename Constraints >
std::vector<carl::Variable> smtrat::mcsat::variableordering::greedy_max_univariate (
            const Constraints & c )
```

14.61.1.6 operator<<()

```
std::ostream& smtrat::mcsat::variableordering::operator<< (
            std::ostream & os,
            const VariableIDs & vids )  [inline]
```

14.62 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)

14.62.1 Function Documentation

14.62.1.1 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 )
```

14.62.1.2 countUnivariates()

```
long smtrat::mcsat::variableordering::detail::countUnivariates (
    const std::vector< carl::Bitset > & constraints,
    std::size_t id ) [inline]
```

14.62.1.3 findMax()

```
carl::Variable smtrat::mcsat::variableordering::detail::findMax (
    const std::vector< carl::Bitset > & constraints,
    const VariableIDs & vids ) [inline]
```

14.62.1.4 max_coefficient()

```
template<typename Constraints >
double smtrat::mcsat::variableordering::detail::max_coefficient (
    const Constraints & constraints,
    carl::Variable v )
```

14.62.1.5 max_degree()

```
template<typename Constraints >
double smtrat::mcsat::variableordering::detail::max_degree (
    const Constraints & constraints,
    carl::Variable v )
```

The maximum degree of this variable.

14.62.1.6 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.

14.62.1.7 num_occurrences()

```
template<typename Constraints >
double smtrat::mcsat::variableordering::detail::num_occurrences (
    const Constraints & constraints,
    carl::Variable v )
```

14.62.1.8 purgeVariable()

```
void smtrat::mcsat::variableordering::detail::purgeVariable (
    std::vector< carl::Bitset > & constraints,
    carl::Variable v,
    const VariableIDs & vids ) [inline]
```

14.62.1.9 stillOccurs()

```
bool smtrat::mcsat::variableordering::detail::stillOccurs (
    const std::vector< carl::Bitset > & constraints,
    std::size_t id ) [inline]
```

14.62.1.10 variablesOf()

```
carl::Bitset smtrat::mcsat::variableordering::detail::variablesOf (
    const ConstraintT & c,
    VariableIDs & vids ) [inline]
```

14.63 smtrat::mcsat::vs Namespace Reference

Namespaces

- helper

Data Structures

- struct DefaultSettings
- struct Explanation
- class ExplanationGenerator

14.64 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 &new↩Substitution)

  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)

14.64.1 Function Documentation

14.64.1.1 addOrMergeTestCandidate()

```
static bool smtrat::mcsat::vs::helper::addOrMergeTest↩
Candidate (
    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.

14.64.1.2 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.

14.64.1.3 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.

14.64.1.4 getFormulaAtoms()

```
void smtrat::mcsat::vs::helper::getFormulaAtoms (
    const FormulaT & f,
    FormulaSetT & result ) [inline]
```

14.64.1.5 substitute()

```
bool smtrat::mcsat::vs::helper::substitute (
    const FormulaT & constr,
    const carl::Variable var,
    const carl::vs::Term< Poly > & term,
    FormulaT & result ) [inline]
```

14.64.1.6 to_formula()

```
FormulaT smtrat::mcsat::vs::helper::to_formula (
    const carl::vs::CaseDistinction< Poly > & docc ) [inline]
```

Converts a DisjunctionOfConstraintConjunctions to a regular Formula.

14.65 smtrat::onecellcad Namespace Reference

Namespaces

- recursive

14.66 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 RAN = carl::RealAlgebraicNumber< smtrat::Rational >
- using RANPoint = carl::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'.

14.66.1 Typedef Documentation

14.66.1.1 MultiCoeffUniPoly

`using smtrat::onecellcad::recursive::MultiCoeffUniPoly = typedef carl::UnivariatePolynomial<MultiPoly>`

14.66.1.2 MultiPoly

`using smtrat::onecellcad::recursive::MultiPoly = typedef carl::Multivariate↩ Polynomial<smtrat::Rational>`

14.66.1.3 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.

14.66.1.4 RAN

`using smtrat::onecellcad::recursive::RAN = typedef carl::RealAlgebraicNumber<smtrat::Rational>`

14.66.1.5 RANMap

`using smtrat::onecellcad::recursive::RANMap = typedef std::map<carl::↩ Variable, RAN>`

14.66.1.6 RANPoint

`using smtrat::onecellcad::recursive::RANPoint = typedef carl::RealAlgebraic↩ Point<smtrat::Rational>`

14.66.1.7 UniPoly

`using smtrat::onecellcad::recursive::UniPoly = typedef carl::Univariate↩ Polynomial<smtrat::Rational>`

14.66.2 Function Documentation

14.66.2.1 createFullspaceCoveringCell()

```
OpenCADCell smtrat::onecellcad::recursive::create↩
FullspaceCoveringCell (
            size_t level )
```

14.66.2.2 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'.

14.66.2.3 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:
- 'poly.gatherVariables()' must be a subset of 'variableOrder'.

14.66.2.4 mergeCellWithPoly()

```
std::optional<OpenCADCell> smtrat::onecellcad::recursive::merge↩
CellWithPoly (
            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)

14.66.2.5 operator<<() [1/3]

```
std::ostream& smtrat::onecellcad::recursive::operator<< (
    std::ostream & o,
    const OpenCADCell & c )
```

14.66.2.6 operator<<() [2/3]

```
std::ostream& smtrat::onecellcad::recursive::operator<< (
    std::ostream & o,
    const Section & s )
```

14.66.2.7 operator<<() [3/3]

```
std::ostream& smtrat::onecellcad::recursive::operator<< (
    std::ostream & o,
    const Sector & s )
```

14.66.2.8 toStdMap()

```
std::map<carl::Variable, RAN> smtrat::onecellcad::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'.

14.67 smtrat::options_detail Namespace Reference

Functions

- void print_cmake_options ()
- void print_info ()
- void print_license ()
- void print_version ()

14.67.1 Function Documentation

14.67.1.1 print_cmake_options()

```
void smtrat::options_detail::print_cmake_options ( )
```

14.67.1.2 print_info()

```
void smtrat::options_detail::print_info ( )
```

14.67.1.3 print_license()

```
void smtrat::options_detail::print_license ( )
```

14.67.1.4 print_version()

```
void smtrat::options_detail::print_version ( )
```

14.68 smtrat::parseformula Namespace Reference

Data Structures

- class FormulaCollector

14.69 smtrat::parser Namespace Reference

Namespaces

- arithmetic
- conversion
- core
- types
- uninterpreted

Data Structures

- 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 AttributeParser
- struct AttributeValueParser
- struct BinaryParser

  Parses binaries: `#b[01]+`

- struct BitvectorTheory

  Implements the theory of bitvectors.

- struct BooleanEncodingTheory

  Implements the theory of bitvectors.

- struct CoreTheory

  Implements the core theory of the booleans.

- struct DecimalParser

  Parses decimals: `numeral.0*numeral`

- struct ErrorHandler
- struct FixedWidthConstant

  Represents a constant of a fixed width.

- struct FunctionInstantiator
- struct HexadecimalParser

  Parses hexadecimals: `#x[0-9a-fA-F]+`

- struct Identifier
- struct IdentifierParser
- struct IndexedFunctionInstantiator
- struct Instantiator
- class InstructionHandler
- struct KeywordParser

  Parses keywords: `:simple_symbol`

- struct LogicParser
- struct NumeralParser

  Parses numerals: `(0 | [1-9][0-9]*)`

- class OutputWrapper
- struct ParserSettings
- struct ParserState
- struct QEParser
- struct QualifiedIdentifierParser
- struct QuantifierParser
- struct RationalPolicies

  Specialization of qi::real_policies for a Rational.

- struct ScriptParser
- struct SExpressionParser

- struct SExpressionSequence
- struct SimpleSymbolParser

  Parses symbols: `simple_symbol | quoted_symbol` *where.*
- struct Skipper
- class SMTLIBParser
- struct SortedVariableParser
- struct SortParser
- struct SpecConstantParser
- struct StringParser

  *Parses strings: " . + " with escape sequences \\ " and *
- struct SymbolParser
- struct TermParser
- struct Theories

  The Theories class combines all implemented theories and provides a single interface to interact with all theories at once.
- struct TheoryError
- struct UninterpretedTheory

  Implements the theory of equalities and uninterpreted functions.
- struct UserFunctionInstantiator

Typedefs
- typedef boost::spirit::istream_iterator BaseIteratorType
- typedef boost::spirit::line_pos_iterator< BaseIteratorType > PositionIteratorType
- typedef PositionIteratorType Iterator
- template<typename T >
  using SExpression = boost::variant< T, boost::recursive_wrapper< SExpressionSequence< T > >>

Enumerations
- enum OptimizationType { OptimizationType::Maximize, OptimizationType::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)

14.69.1 Typedef Documentation

14.69.1.1 BaseIteratorType `typedef boost::spirit::istream_iterator smtrat::parser::BaseIteratorType`

14.69.1.2 Iterator `typedef PositionIteratorType smtrat::parser::Iterator`

14.69.1.3 PositionIteratorType `typedef boost::spirit::line_pos_iterator<BaseIteratorType> smtrat::parser::Posit`

14.69.1.4 SExpression `template<typename T >`

```
using smtrat::parser::SExpression = typedef boost::variant<T, boost::recursive_wrapper<SExpressionSequence<T>
>>
```

14.69.2 Enumeration Type Documentation

14.69.2.1 OptimizationType `enum smtrat::parser::OptimizationType [strong]`

Enumerator

Maximize	
Minimize	

14.69.3 Function Documentation

14.69.3.1 at_most_k()

```
FormulaT smtrat::parser::at_most_k (
    size_t k,
    const std::vector< FormulaT > & set )
```

14.69.3.2 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 )
```

14.69.3.3 operator<<() [1/5]

```
std::ostream& smtrat::parser::operator<< (
    std::ostream & os,
    const Attribute & attr )  [inline]
```

14.69.3.4 operator<<() [2/5]

```
template<typename T >
std::ostream& smtrat::parser::operator<< (
    std::ostream & os,
    const FixedWidthConstant< T > & fwc )  [inline]
```

14.69.3.5 operator<<() [3/5]

```
std::ostream& smtrat::parser::operator<< (
    std::ostream & os,
    const Identifier & identifier )  [inline]
```

14.69.3.6 operator<<() [4/5]

```
template<typename T >
std::ostream& smtrat::parser::operator<< (
    std::ostream & os,
    const SExpressionSequence< T > & ses )  [inline]
```

14.69.3.7 operator<<() [5/5]

```
std::ostream& smtrat::parser::operator<< (
    std::ostream & os,
    OptimizationType ot )  [inline]
```

14.69.3.8 registerParserSettings()

```
template<typename T >
void smtrat::parser::registerParserSettings (
    T & parser )
```

14.70 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)

14.70.1 Typedef Documentation

14.70.1.1 OperatorType

```
using smtrat::parser::arithmetic::OperatorType = typedef boost::variant<Poly↩
::ConstructorOperation, carl::Relation>
```

14.70.2 Function Documentation

14.70.2.1 isBooleanIdentity()

```
bool smtrat::parser::arithmetic::isBooleanIdentity (
    const OperatorType & op,
    const std::vector< types::TermType > & arguments,
    TheoryError & errors )  [inline]
```

14.70.2.2 makeConstraint()

```
FormulaT smtrat::parser::arithmetic::makeConstraint (
    ArithmeticTheory & at,
    const Poly & lhs,
    const Poly & rhs,
    carl::Relation rel )  [inline]
```

14.71 smtrat::parser::conversion Namespace Reference

Data Structures

- struct Converter
- struct Converter< FormulaT >
- struct Converter< Poly >
- struct Converter< types::BVTerm >
- 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.

14.72 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)

14.72.1 Function Documentation

14.72.1.1 convertArguments()

```
bool smtrat::parser::core::convertArguments (
    const std::vector< types::TermType > & arguments,
    std::vector< FormulaT > & result,
    TheoryError & errors ) [inline]
```

14.72.1.2 convertTerm()

```
bool smtrat::parser::core::convertTerm (
    const types::TermType & term,
    FormulaT & result ) [inline]
```

14.73 smtrat::parser::types Namespace Reference

Data Structures

- struct ArithmeticTheory

  Types of the arithmetic theory.
- struct BitvectorTheory

  Types of the theory of bitvectors.
- struct CoreTheory

  Types of the core theory.
- 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.

14.73.1 Typedef Documentation

14.73.1.1 AttributeTypes

`typedef carl::mpl_concatenate< TermTypes, boost::mpl::vector< SExpressionSequence<typ std::string, bool, boost::spirit::unused_type > >::type smtrat::parser::types::AttributeTypes`

List of all types of attributes.

14.73.1.2 AttributeValue

`typedef carl::mpl_variant_of<AttributeTypes>::type smtrat::parser::types::AttributeVal`

Variant type for all attributes.

14.73.1.3 BVConstraint

`typedef carl::BVConstraint smtrat::parser::types::BVConstraint`

Typedef for bitvector constraint.

14.73.1.4 BVTerm

`typedef carl::BVTerm smtrat::parser::types::BVTerm`

Typedef for bitvector term.

14.73.1.5 BVVariable

`typedef carl::BVVariable smtrat::parser::types::BVVariable`

14.73.1.6 ConstType

`typedef carl::mpl_variant_of<ConstTypes>::type smtrat::parser::types::ConstType`

Variant type for all constants.

14.73.1.7 ConstTypes

`typedef carl::mpl_concatenate< CoreTheory::ConstTypes, >::type smtrat::parser::types::Co`

List of all types of constants.

14.73.1.8 ExpressionType

`typedef carl::mpl_variant_of<ExpressionTypes>::type smtrat::parser::types::Expressio`

Variant type for all expressions.

14.73.1.9 ExpressionTypes

`typedef carl::mpl_concatenate< CoreTheory::ExpressionTypes, >::type smtrat::parser::types::ExpressionTypes`

List of all types of expressions.

14.73.1.10 TermType `typedef carl::mpl_variant_of<TermTypes>::type smtrat::parser::types::TermType`

Variant type for all terms.

14.73.1.11 TermTypes `typedef carl::mpl_concatenate< CoreTheory::TermTypes, >::type smtrat::parser::types::Te`

List of all types of terms.

14.73.1.12 UTerm `using smtrat::parser::types::UTerm = typedef carl::UTerm`

14.73.1.13 VariableType `typedef carl::mpl_variant_of<VariableTypes>::type smtrat::parser::types::VariableType`

Variant type for all variables.

14.73.1.14 VariableTypes `typedef carl::mpl_concatenate< CoreTheory::VariableTypes, >::type smtrat::parser::types::VariableTypes`

List of all types of variables.

14.74 smtrat::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)

14.74.1 Function Documentation

14.74.1.1 convertArguments()

```
bool smtrat::parser::uninterpreted::convertArguments (
    const std::vector< types::TermType > & arguments,
    std::vector< types::UTerm > & result,
    TheoryError & errors ) [inline]
```

14.74.1.2 convertTerm()

```
bool smtrat::parser::uninterpreted::convertTerm (
    const types::TermType & term,
    types::UTerm & result ) [inline]
```

14.75 smtrat::preprocessor Namespace Reference

14.76 smtrat::qe Namespace Reference

Namespaces
- cad
- fm

Typedefs
- using QEQuery = std::vector< std::pair< QuantifierType, std::vector< carl::Variable > >>

Enumerations
- enum QuantifierType { QuantifierType::EXISTS, QuantifierType::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)

14.76.1 Typedef Documentation

14.76.1.1 QEQuery `using smtrat::qe::QEQuery = typedef std::vector<std::pair<QuantifierType,std↩ ::vector<carl::Variable> >>`

14.76.2 Enumeration Type Documentation

14.76.2.1 QuantifierType `enum smtrat::qe::QuantifierType [strong]`

Enumerator

EXISTS	
FORALL	

14.76.3 Function Documentation

14.76.3.1 eliminateQuantifiers()

```
FormulaT smtrat::qe::eliminateQuantifiers (
    const FormulaT & qfree,
    const QEQuery & quantifiers ) [inline]
```

14.76.3.2 flattenQEQuery()

```
std::vector<std::pair<QuantifierType,carl::Variable> > smtrat::qe↩
::flattenQEQuery (
    const QEQuery & query ) [inline]
```

14.76.3.3 operator<<() [1/2]

```
std::ostream& smtrat::qe::operator<< (
    std::ostream & os,
    const QEQuery & q ) [inline]
```

14.76.3.4 operator<<() [2/2]

```
std::ostream& smtrat::qe::operator<< (
    std::ostream & os,
    QuantifierType qt ) [inline]
```

14.77 smtrat::qe::cad Namespace Reference

Data Structures

- class CAD
- class CADElimination
- struct CADSettings
- 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)

14.77.1 Function Documentation

14.77.1.1 eliminateQuantifiers()

```
FormulaT smtrat::qe::cad::eliminateQuantifiers (
    const FormulaT & qfree,
    const QEQuery & quantifiers )
```

14.77.1.2 flip_map()

```
template<typename key , typename value >
std::multimap<value, key> smtrat::qe::cad::flip_map (
    const std::map< key, value > & src )
```

14.77.1.3 operator<<()

```
template<typename S >
std::ostream& smtrat::qe::cad::operator<< (
            std::ostream & os,
            const Projection< S > & p )
```

14.78 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)

14.78.1 Function Documentation

14.78.1.1 eliminateQuantifiers()

```
FormulaT smtrat::qe::fm::eliminateQuantifiers (
            const FormulaT & qfree,
            const QEQuery & quantifiers )
```

14.79 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)

14.79.1 Function Documentation

14.79.1.1 getCPULimit()

```
std::size_t smtrat::resource::getCPULimit ( ) [inline]
```

14.79.1.2 getMemoryLimit()

```
std::size_t smtrat::resource::getMemoryLimit ( ) [inline]
```

14.79.1.3 installSignalHandler()

```
void smtrat::resource::installSignalHandler ( ) [inline]
```

14.79.1.4 setCPULimit()

```
void smtrat::resource::setCPULimit (
            std::size_t ) [inline]
```

14.79.1.5 setMemoryLimit()

```
void smtrat::resource::setMemoryLimit (
            std::size_t ) [inline]
```

14.79.1.6 signalHandler()

```
void smtrat::resource::signalHandler (
    int signal ) [inline]
```

14.79.1.7 usedCPU()

```
std::size_t smtrat::resource::usedCPU ( ) [inline]
```

14.80 smtrat::sat Namespace Reference

Namespaces

- detail

14.81 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)

14.81.1 Function Documentation

14.81.1.1 validateClause() [1/2]

```
template<typename T >
void smtrat::sat::detail::validateClause (
    const T & t,
    bool enabled )
```

14.81.1.2 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 )
```

14.82 smtrat::settings Namespace Reference

Data Structures

- struct CoreSettings
- struct ModuleSettings
- struct Settings
- struct SolverSettings

14.83 smtrat::statistics Namespace Reference

Data Structures

- struct StatisticsSettings

Functions

- template<typename T >
  void registerStatisticsSettings (T &parser)

14.83.1 Function Documentation

14.83.1.1 registerStatisticsSettings()

```
template<typename T >
void smtrat::statistics::registerStatisticsSettings (
            T & parser )
```

14.84 smtrat::uf_impl Namespace Reference

Data Structures

- 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 >
- 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.
- 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.

14.85 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.

14.85.1 Detailed Description

Contains strategy implementations for unsat core computations.

14.86 smtrat::validation Namespace Reference

Data Structures

- class ValidationCollector
- class ValidationPoint
- struct ValidationPrinter
- struct ValidationSettings

Enumerations

- enum ValidationOutputFormat { ValidationOutputFormat::SMTLIB }

Functions

- auto & get (const std::string &channel, const std::string &name)
- 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)

14.86.1 Enumeration Type Documentation

14.86.1.1 ValidationOutputFormat

```
enum smtrat::validation::ValidationOutputFormat [strong]
```

Enumerator

SMTLIB	

14.86.2 Function Documentation

14.86.2.1 get()

```
auto& smtrat::validation::get (
    const std::string & channel,
    const std::string & name ) [inline]
```

14.86.2.2 operator<<() [1/2]

```
template<ValidationOutputFormat SOF>
std::ostream& smtrat::validation::operator<< (
    std::ostream & os,
    ValidationPrinter< SOF > )
```

14.86.2.3 operator<<() [2/2]

```
template<>
std::ostream& smtrat::validation::operator<< (
    std::ostream & os,
    ValidationPrinter< ValidationOutputFormat::SMTLIB > )
```

14.86.2.4 registerValidationSettings()

```
template<typename T >
void smtrat::validation::registerValidationSettings (
    T & parser )
```

14.86.2.5 validation_formulas_as_smtlib()

```
auto smtrat::validation::validation_formulas_as_smtlib (
)
```

14.86.2.6 validation_formulas_to_smtlib_file()

```
void smtrat::validation::validation_formulas_to_↩
smtlib_file (
    const std::string & filename )
```

14.87 smtrat::vb Namespace Reference

Data Structures

- class Bound

  Class for the bound of a variable.

- class Variable

  Class for 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.

14.87.1 Function Documentation

14.87.1.1 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

_os	The stream to print on.
_vs	The variable bounds manager to print.

Returns The stream after printing on it.

14.88 smtrat::vs Namespace Reference

Data Structures

- class Condition
- class State
- class Substitution
- struct substitutionPointerEqual
- struct substitutionPointerHash
- struct unsignedTripleCmp

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 smtrat::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 smtrat::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 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.

- void splitSosDecompositions (DisjunctionOfConstraintConjunctions &_toSimplify)
- DisjunctionOfConstraintConjunctions getSignCombinations (const smtrat::ConstraintT &)

  For a given constraint f_1...*f_n ~ 0 this method computes all combinations of constraints f_1 ~_1 0 ...*

- 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 smtrat::ConstraintT &_cons, const Substitution &_subs, DisjunctionOfConstraintConjunctions &_result, bool _accordingPaper, Variables &_conflictingVariables, const smtrat::EvalDoubleIntervalMap &_solutionSpace)
- bool substituteNormal (const smtrat::ConstraintT &_cons, const Substitution &_subs, DisjunctionOfConstraintConjunctions &_result, bool _accordingPaper, Variables &_conflictingVariables, const smtrat::EvalDoubleIntervalMap &_solutionSpace)
- bool 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.

- bool 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.

- bool 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.

- bool 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.

- bool substitutePlusEps (const smtrat::ConstraintT &_cons, const Substitution &_subs, DisjunctionOfConstraintConjunctions &_result, bool _accordingPaper, Variables &_conflictingVariables, const smtrat::EvalDoubleIntervalMap &_solutionSpace)
- bool substituteEpsGradients (const smtrat::ConstraintT &_cons, const Substitution &_subs, const Relation _relation, DisjunctionOfConstraintConjunctions &_result, bool _accordingPaper, Variables &_conflictingVariables, const smtrat::EvalDoubleIntervalMap &_solutionSpace)
- void substituteInf (const smtrat::ConstraintT &_cons, const Substitution &_subs, DisjunctionOfConstraintConjunctions &_result, Variables &_conflictingVariables, const smtrat::EvalDoubleIntervalMap &_solutionSpace)
- void substituteInfLessGreater (const smtrat::ConstraintT &_cons, const Substitution &_subs, DisjunctionOfConstraintConjunction &_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 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.

- void 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.

- 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 smtrat::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 smtrat::ConstraintT &, const Substitution &, DisjunctionOfConstraintConjunctions &, bool _accordingPaper, carl::Variables &, const smtrat::EvalDoubleIntervalMap &)

  Applies a substitution to a constraint and stores the results in the given vector.
- bool 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.
- bool 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 -> t+epsilon] with x as the variable and c and b polynomials in the real theory excluding x.
- bool substituteEpsGradients (const smtrat::ConstraintT &_cons, const Substitution &_subs, const carl↩ ::Relation _relation, DisjunctionOfConstraintConjunctions &, bool _accordingPaper, carl::Variables &_↩ conflictingVariables, const smtrat::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 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 -> -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.

14.88.1 Typedef Documentation

14.88.1.1 ConditionList `typedef std::list< const Condition* > smtrat::vs::ConditionList`

14.88.1.2 ConditionSetSet `typedef std::set< carl::PointerSet<Condition> > smtrat::vs::ConditionSetSet`

14.88.1.3 ConditionSetSetSet `typedef std::set< ConditionSetSet > smtrat::vs::ConditionSetSetSet`

14.88.1.4 ConstraintVector `typedef std::vector<smtrat::ConstraintT> smtrat::vs::ConstraintVector`

a vector of constraints

14.88.1.5 DisjunctionOfConditionConjunctions `typedef std::vector< ConditionList > smtrat::vs::DisjunctionOfCon`

14.88.1.6 DisjunctionOfConstraintConjunctions `typedef std::vector<ConstraintVector> smtrat::vs::DisjunctionOfC`

a vector of vectors of constraints

14.88.1.7 SubstitutionFastPointerMap

```
template<typename T >
using smtrat::vs::SubstitutionFastPointerMap = typedef std::unordered_map<const smtrat::Poly*,
T, substitutionPointerHash, substitutionPointerEqual>
```

14.88.1.8 UnsignedTriple

```
typedef std::pair<size_t, std::pair<size_t, size_t> > smtrat::vs::UnsignedTriple
```

14.88.1.9 ValStatePair

```
typedef std::pair<UnsignedTriple, vs::State*> smtrat::vs::ValStatePair
```

14.88.1.10 ValuationMap

```
typedef std::map<UnsignedTriple, vs::State*, unsignedTripleCmp> smtrat::vs::Valuation
```

14.88.2 Function Documentation

14.88.2.1 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

_toCombine	The vectors to combine.
_combination	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.

14.88.2.2 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.

14.88.2.3 getEvenBitStrings() [2/2]

```
void smtrat::vs::getEvenBitStrings (
            std::size_t _length,
            std::vector< std::bitset< MAX_PRODUCT_SPLIT_NUMBER > > & _strings )
```

14.88.2.4 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.

14.88.2.5 getOddBitStrings() [2/2]

```
void smtrat::vs::getOddBitStrings (
    std::size_t _length,
    std::vector< std::bitset< MAX_PRODUCT_SPLIT_NUMBER > > & _strings )
```

14.88.2.6 getSignCombinations()

```
DisjunctionOfConstraintConjunctions smtrat::vs::getSign↩
Combinations (
    const smtrat::ConstraintT &  )
```

For a given constraint f_1∗...∗f_n ∼ 0 this method computes all combinations of constraints f_1 ∼_1 0 ...

f_n ∼_n 0 such that

```
f_1 ~_1 0 and ... and f_n ~_n 0   iff   f_1*...*f_n ~ 0
```

holds.

Parameters

_constraint	A pointer to the constraint to split this way.

Returns The resulting combinations.

14.88.2.7 operator<<() [1/2]

```
std::ostream& smtrat::vs::operator<< (
    std::ostream & _out,
    const Condition & _condition )
```

14.88.2.8 operator<<() [2/2]

```
std::ostream & smtrat::vs::operator<< (
    std::ostream & os,
    const Substitution & s )
```

Prints the given substitution on the given output stream.

Parameters

_out	The output stream, on which to write.
_substitution	The substitution to print.

Returns The output stream after printing the substitution on it.

14.88.2.9 print()

```
void smtrat::vs::print (
    DisjunctionOfConstraintConjunctions & _substitutionResults )
```

Prints the given disjunction of conjunction of constraints.

Parameters

_substitutionResults	The disjunction of conjunction of constraints to print.

14.88.2.10 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

_toSimplify	The disjunction of conjunctions to simplify.

14.88.2.11 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

_toSimplify	The disjunction of conjunctions to simplify.
_conflictingVars	
_solutionSpace	

14.88.2.12 simplify() [3/3]

```
void smtrat::vs::simplify (
    DisjunctionOfConstraintConjunctions & _toSimplify,
    Variables & _conflictingVars,
    const smtrat::EvalDoubleIntervalMap & _solutionSpace )
```

14.88.2.13 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

_toSimplify	The conjunction of the constraints to split.
_result	The result, being a disjunction of conjunctions of constraints.
_onlyNeq	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.

14.88.2.14 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

_constraint	A pointer to the constraint to split.
_onlyNeq	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.

14.88.2.15 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

_toSimplify	The disjunction of conjunctions of the constraints to split.
_onlyNeq	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.

14.88.2.16 splitSosDecompositions()

```
void smtrat::vs::splitSosDecompositions (
    DisjunctionOfConstraintConjunctions & _toSimplify )
```

14.88.2.17 substitute() [1/2]

```
bool smtrat::vs::substitute (
    const smtrat::ConstraintT & ,
    const Substitution & ,
    DisjunctionOfConstraintConjunctions & ,
    bool _accordingPaper,
    carl::Variables & ,
    const smtrat::EvalDoubleIntervalMap &  )
```

Applies a substitution to a constraint and stores the results in the given vector.

Parameters

_cons	The constraint to substitute in.
_subs	The substitution to apply.
_result	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.

14.88.2.18 substitute() [2/2]

```
bool smtrat::vs::substitute (
    const smtrat::ConstraintT & _cons,
    const Substitution & _subs,
    DisjunctionOfConstraintConjunctions & _result,
    bool _accordingPaper,
    Variables & _conflictingVariables,
    const smtrat::EvalDoubleIntervalMap & _solutionSpace )
```

14.88.2.19 substituteEpsGradients() [1/2]

```
bool smtrat::vs::substituteEpsGradients (
    const smtrat::ConstraintT & _cons,
    const Substitution & _subs,
    const carl::Relation _relation,
    DisjunctionOfConstraintConjunctions & ,
    bool _accordingPaper,
    carl::Variables & _conflictingVariables,
    const smtrat::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

_cons	The constraint to substitute in.
_subs	The substitution to apply.

Parameters

_relation	The relation symbol, deciding whether the substitution result must be negative or positive.
_result	The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints.
_accordingPaper	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).
_conflictingVariables	If a conflict with the given solution space occurs, the variables being part of this conflict are stored in this container.
_solutionSpace	The solution space in form of double intervals of the variables occurring in the given constraint.

14.88.2.20 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 )
```

14.88.2.21 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 -> -infinity] 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 {=,!=,<,>,<=,>=} and k as the maximum degree of x in f.

Parameters

_cons	The constraint to substitute in.
_subs	The substitution to apply.
_result	The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints.
_conflictingVariables	If a conflict with the given solution space occurs, the variables being part of this conflict are stored in this container.
_solutionSpace	The solution space in form of double intervals of the variables occurring in the given constraint.

14.88.2.22 substituteInf() [2/2]

```
void smtrat::vs::substituteInf (
    const smtrat::ConstraintT & _cons,
    const Substitution & _subs,
    DisjunctionOfConstraintConjunctions & _result,
    Variables & _conflictingVariables,
    const smtrat::EvalDoubleIntervalMap & _solutionSpace )
```

14.88.2.23 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 -> +/-infinity] 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

_cons	The constraint to substitute in.
_subs	The substitution to apply.
_result	The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints.

14.88.2.24 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

_cons	The constraint to substitute in.
_subs	The substitution to apply.
_result	The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints.
_accordingPaper	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).
_conflictingVariables	If a conflict with the given solution space occurs, the variables being part of this conflict are stored in this container.
_solutionSpace	The solution space in form of double intervals of the variables occurring in the given constraint.

14.88.2.25 substituteNormal() [2/2]

```
bool smtrat::vs::substituteNormal (
    const smtrat::ConstraintT & _cons,
    const Substitution & _subs,
    DisjunctionOfConstraintConjunctions & _result,
    bool _accordingPaper,
    Variables & _conflictingVariables,
    const smtrat::EvalDoubleIntervalMap & _solutionSpace )
```

14.88.2.26 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

_radicand	The radicand of the square root.
_q	The summand not containing the square root.
_r	The coefficient of the radicand.
_result	The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints.
_accordingPaper	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).

14.88.2.27 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

_radicand	The radicand of the square root.
_q	The summand not containing the square root.
_r	The coefficient of the radicand.
_s	The denominator of the expression containing the square root.
_result	The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints.
_accordingPaper	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).

14.88.2.28 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

_radicand	The radicand of the square root.
_q	The summand not containing the square root.
_r	The coefficient of the radicand.
_s	The denominator of the expression containing the square root.
_result	The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints.
_accordingPaper	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).

14.88.2.29 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

_radicand	The radicand of the square root.
_q	The summand not containing the square root.
_r	The coefficient of the radicand.
_result	The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints.
_accordingPaper	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).

14.88.2.30 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

_cons	The constraint to substitute in.
_subs	The substitution to apply.
_result	The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints.

14.88.2.31 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 -> 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 {=,!=,<,>,<=,>=} and k as the maximum degree of x in f.

Parameters

_cons	The constraint to substitute in.
_subs	The substitution to apply.
_result	The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints.
_accordingPaper	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).
_conflictingVariables	If a conflict with the given solution space occurs, the variables being part of this conflict are stored in this container.
_solutionSpace	The solution space in form of double intervals of the variables occurring in the given constraint.

14.88.2.32 substitutePlusEps() [2/2]

```
bool smtrat::vs::substitutePlusEps (
    const smtrat::ConstraintT & _cons,
    const Substitution & _subs,
    DisjunctionOfConstraintConjunctions & _result,
    bool _accordingPaper,
    Variables & _conflictingVariables,
    const smtrat::EvalDoubleIntervalMap & _solutionSpace )
```

14.88.2.33 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

_cons	The constraint to substitute in.
_subs	The substitution to apply.
_result	The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints.

15 Data Structure Documentation

15.1 smtrat::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

15.1.1 Constructor & Destructor Documentation

15.1.1.1 ~AbstractModuleFactory()

```
virtual smtrat::AbstractModuleFactory::~AbstractModuleFactory ( ) [virtual], [default]
```

15.1.2 Member Function Documentation

15.1.2.1 create()

```
virtual Module* smtrat::AbstractModuleFactory::create (
    const ModuleInput * _formula,
    Conditionals & _conditionals,
    Manager *const _manager ) [pure virtual]
```

Implemented in smtrat::ModuleFactory< Module >.

15.1.2.2 moduleName()

```
virtual std::string smtrat::AbstractModuleFactory::moduleName ( ) const [pure virtual]
```

Implemented in smtrat::ModuleFactory< Module >.

15.2 smtrat::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

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 datastructures.

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.

15.2.2 Constructor & Destructor Documentation

15.2.2.1 AbstractTheory()

```
smtrat::parser::AbstractTheory::AbstractTheory (
    ParserState * state ) [inline]
```

15.2.2.2 ~AbstractTheory()

```
virtual smtrat::parser::AbstractTheory::~AbstractTheory ( ) [inline], [virtual]
```

15.2.3 Member Function Documentation

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.

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.

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::ArithmeticTheory, smtrat::parser::CoreTheory, and smtrat::parser::BitvectorTheory.

15.2.3.4 expandDistinct()

```
template<typename T , typename Builder >
FormulaT smtrat::parser::AbstractTheory::expandDistinct (
            const std::vector< T > & values,
            const Builder & neqBuilder )  [inline]
```

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::BitvectorTheory, smtrat::parser::ArithmeticTheory, and smtrat::parser::CoreTheory.

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::ArithmeticTheory, smtrat::parser::BitvectorTheory, smtrat::parser::CoreTheory, and smtrat::parser::UninterpretedTheory.

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::ArithmeticTheory, smtrat::parser::BitvectorTheory, and smtrat::parser::CoreTheory.

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.

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.

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.

15.2.4 Field Documentation

15.2.4.1 state

```
ParserState* smtrat::parser::AbstractTheory::state
```

15.3 smtrat::cad::sample_compare::absvalue Struct Reference

```
#include <SampleComparator.h>
```

15.4 smtrat::analyzer::AnalysisSettings Struct Reference

```
#include <settings.h>
```

Data Fields

- bool enabled = false
- std::string analyze_projections = "none"

15.4.1 Field Documentation

15.4.1.1 analyze_projections

```
std::string smtrat::analyzer::AnalysisSettings::analyze_projections
= "none"
```

15.4.1.2 enabled

```
bool smtrat::analyzer::AnalysisSettings::enabled = false
```

15.5 smtrat::analyzer::AnalyzerStatistics Struct Reference

```
#include <statistics.h>
```

Public Member Functions

- template<typename T >
  void add (const std::string &key, const T &value)

15.5.1 Member Function Documentation

15.5.1.1 add()

```
template<typename T >
void smtrat::analyzer::AnalyzerStatistics::add (
            const std::string & key,
            const T & value ) [inline]
```

15.6 smtrat::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)

15.6.1 Constructor & Destructor Documentation

15.6.1.1 AnnotatedBVTerm() [1/4]

```
smtrat::AnnotatedBVTerm::AnnotatedBVTerm ( ) [inline]
```

15.6.1.2 AnnotatedBVTerm() [2/4]

```
smtrat::AnnotatedBVTerm::AnnotatedBVTerm (
            const BVAnnotation & _type,
            const carl::BVTerm & _term ) [inline]
```

15.6.1.3 AnnotatedBVTerm() [3/4]

```
smtrat::AnnotatedBVTerm::AnnotatedBVTerm (
            std::size_t _width,
            bool _signed,
            Integer _offset = 0 ) [inline]
```

15.6.1.4 AnnotatedBVTerm() [4/4]

```
smtrat::AnnotatedBVTerm::AnnotatedBVTerm (
            const BVAnnotation & _type ) [inline]
```

15.6.2 Member Function Documentation

15.6.2.1 operator()()

```
const carl::BVTerm& smtrat::AnnotatedBVTerm::operator() ( ) const [inline]
```

15.6.2.2 term()

```
const carl::BVTerm& smtrat::AnnotatedBVTerm::term ( ) const [inline]
```

15.6.2.3 type()

```
const BVAnnotation& smtrat::AnnotatedBVTerm::type ( ) const [inline]
```

15.6.3 Friends And Related Function Documentation

15.6.3.1 operator<< `std::ostream& operator<< ( std::ostream & _out, const AnnotatedBVTerm & _term ) [friend]`

15.7 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.

15.7.1 Detailed Description

All properties needed to archive log files.

15.7.2 Field Documentation

15.7.2.1 filename_archive `std::string benchmax::slurm::ArchiveProperties::filename_archive`

Filename of the archive.

15.7.2.2 tmp_dir `std::string benchmax::slurm::ArchiveProperties::tmp_dir`

Temporary directory to look for output files.

15.8 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)
- 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 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 &result, TheoryError &errors)

  Instantiate a variable within a term.

- 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 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 > &)

  Initialize the global symbol table for constants.

Data Fields

- std::map< std::string, arithmetic::OperatorType > ops
- std::map< carl::Variable, std::tuple< FormulaT, Poly, Poly > > mITEs
- ParserState * state

15.8.1 Detailed Description

Implements the theory of arithmetic, including LRA, LIA, NRA and NIA.

15.8.2 Constructor & Destructor Documentation

15.8.2.1 ArithmeticTheory()

```
smtrat::parser::ArithmeticTheory::ArithmeticTheory (
    ParserState * state )
```

15.8.3 Member Function Documentation

15.8.3.1 addConstants()

```
static void smtrat::parser::AbstractTheory::addConstants (
    qi::symbols< char, types::ConstType > & ) [inline], [static], [inherited]
```

Initialize the global symbol table for constants.

15.8.3.2 addSimpleSorts()

```
void smtrat::parser::ArithmeticTheory::addSimpleSorts (
    qi::symbols< char, carl::Sort > & sorts ) [static]
```

15.8.3.3 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.

15.8.3.4 expandDistinct()

```
template<typename T , typename Builder >
FormulaT smtrat::parser::AbstractTheory::expandDistinct (
    const std::vector< T > & values,
    const Builder & neqBuilder ) [inline], [inherited]
```

15.8.3.5 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.

15.8.3.6 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.

15.8.3.7 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.

15.8.3.8 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.

15.8.3.9 refreshVariable()

```
virtual bool smtrat::parser::AbstractTheory::refreshVariable (
    const types::VariableType & ,
    types::VariableType & ,
    TheoryError & errors )  [inline], [virtual], [inherited]
```

Reimplemented in smtrat::parser::BitvectorTheory.

15.8.3.10 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.

15.8.4 Field Documentation

15.8.4.1 mITEs `std::map<carl::Variable, std::tuple<FormulaT, Poly, Poly> > smtrat::parser::ArithmeticTheory::mITEs`

15.8.4.2 ops `std::map<std::string, arithmetic::OperatorType> smtrat::parser::ArithmeticTheory::ops`

15.8.4.3 state `ParserState* smtrat::parser::AbstractTheory::state [inherited]`

15.9 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 > ExpressionTypes
- typedef mpl::vector< carl::Variable, Rational, FixedWidthConstant< Integer >, Poly > TermTypes
- typedef carl::mpl_variant_of< TermTypes >::type TermType

15.9.1 Detailed Description

Types of the arithmetic theory.

15.9.2 Member Typedef Documentation

15.9.2.1 ConstTypes `typedef mpl::vector<Rational, FixedWidthConstant<Integer> > smtrat::parser::types::Arit`

15.9.2.2 ExpressionTypes `typedef mpl::vector<carl::Variable, Rational, FixedWidthConstant<Integer>, Poly> smtrat::parser::types::ArithmeticTheory::ExpressionTypes`

15.9.2.3 TermType `typedef carl::mpl_variant_of<TermTypes>::type smtrat::parser::types::ArithmeticTheory::Term`

15.9.2.4 TermTypes `typedef mpl::vector<carl::Variable, Rational, FixedWidthConstant<Integer>, Poly> smtrat::parser::types::ArithmeticTheory::TermTypes`

15.9.2.5 VariableTypes `typedef mpl::vector<carl::Variable> smtrat::parser::types::ArithmeticTheory::VariableT`

15.10 smtrat::execution::Assertion Struct Reference

`#include <ExecutionState.h>`

Data Fields

- FormulaT formula

15.10.1 Field Documentation

15.10.1.1 formula `FormulaT smtrat::execution::Assertion::formula`

15.11 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)

15.11.1 Member Typedef Documentation

15.11.1.1 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

15.11.2 Constructor & Destructor Documentation

15.11.2.1 AssignmentCollector()

```
smtrat::cad::preprocessor::AssignmentCollector::AssignmentCollector (
            Model & model ) [inline]
```

15.11.3 Member Function Documentation

15.11.3.1 collect()

```
AssignmentCollector::CollectionResult smtrat::cad::preprocessor::AssignmentCollector::collect (
            std::map< ConstraintT, ConstraintT > & constraints )
```

15.11.3.2 constraints() [1/2]

```
auto& smtrat::cad::preprocessor::AssignmentCollector::constraints ( ) [inline]
```

15.11.3.3 constraints() [2/2]

```
const auto& smtrat::cad::preprocessor::AssignmentCollector::constraints ( ) const [inline]
```

15.11.3.4 reasons() [1/2]

```
auto& smtrat::cad::preprocessor::AssignmentCollector::reasons ( ) [inline]
```

15.11.3.5 reasons() [2/2]

```
const auto& smtrat::cad::preprocessor::AssignmentCollector::reasons ( ) const [inline]
```

15.12 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

15.12.1 Member Function Documentation

15.12.1.1 active()

```
bool smtrat::mcsat::arithmetic::AssignmentFinder::active (
    const mcsat::Bookkeeping & data,
    const FormulaT & f ) const
```

15.12.1.2 operator()()

```
std::optional< AssignmentOrConflict > smtrat::mcsat::arithmetic::↩
AssignmentFinder::operator() (
    const mcsat::Bookkeeping & data,
    carl::Variable var ) const
```

15.13 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

15.13.1 Member Function Documentation

15.13.1.1 active()

```
template<class Settings >
bool smtrat::mcsat::smtaf::AssignmentFinder< Settings >::active (
    const mcsat::Bookkeeping & ,
    const FormulaT &  ) const  [inline]
```

15.13.1.2 operator()()

```
template<class Settings >
std::optional<AssignmentOrConflict> smtrat::mcsat::smtaf::AssignmentFinder< Settings >↩
::operator() (
    const mcsat::Bookkeeping & data,
    carl::Variable var ) const  [inline]
```

15.14 smtrat::mcsat::arithmetic::AssignmentFinder_detail Class Reference

```
#include <AssignmentFinder_arithmetic.h>
```

Public Types

- using RAN = carl::RealAlgebraicNumber< Rational >

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 ()

15.14.1 Member Typedef Documentation

15.14.1.1 RAN

```
using smtrat::mcsat::arithmetic::AssignmentFinder_detail::RAN = carl::Real↩
AlgebraicNumber<Rational>
```

15.14.2 Constructor & Destructor Documentation

15.14.2.1 AssignmentFinder_detail()

```
smtrat::mcsat::arithmetic::AssignmentFinder_detail::Assignment↩
Finder_detail (
            carl::Variable var,
            const Model & model )  [inline]
```

15.14.3 Member Function Documentation

15.14.3.1 addConstraint()

```
bool smtrat::mcsat::arithmetic::AssignmentFinder_detail::addConstraint
(
            const FormulaT & f )  [inline]
```

15.14.3.2 addMVBound()

```
bool smtrat::mcsat::arithmetic::AssignmentFinder_detail::addMVBound (
            const FormulaT & f )  [inline]
```

15.14.3.3 computeCover()

```
Covering smtrat::mcsat::arithmetic::AssignmentFinder_detail::compute↩
Cover ( )  [inline]
```

15.14.3.4 findAssignment()

```
AssignmentOrConflict smtrat::mcsat::arithmetic::AssignmentFinder_↩
detail::findAssignment ( )  [inline]
```

15.15 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

15.15.1 Constructor & Destructor Documentation

15.15.1.1 AssignmentFinder_SMT()

```
smtrat::mcsat::smtaf::AssignmentFinder_SMT::Assignment↩
Finder_SMT (
            VariableRange variables,
            const Model & model )  [inline]
```

15.15.2 Member Function Documentation

15.15.2.1 addConstraint()

```
boost::tribool smtrat::mcsat::smtaf::AssignmentFinder_SMT::add↩
Constraint (
            const FormulaT & f )
```

15.15.2.2 addMVBound()

```
boost::tribool smtrat::mcsat::smtaf::AssignmentFinder_SMT::addMVBound
(
            const FormulaT & f )
```

15.15.2.3 findAssignment() [1/2]

```
std::optional< AssignmentOrConflict > smtrat::mcsat::smtaf↩
::AssignmentFinder_SMT::findAssignment ( ) const
```

15.15.2.4 findAssignment() [2/2]

```
std::optional< AssignmentOrConflict > smtrat::mcsat::smtaf↩
::AssignmentFinder_SMT::findAssignment (
            const VariablePos excludeVar ) const
```

15.16 smtrat::cadcells::datastructures::detail::AssignmentProperties Struct Reference

```
#include <projections.h>
```

Data Fields

- std::map< PolyRef, carl::ran::real_roots_result< RAN > > real_roots
- std::map< PolyRef, bool > is_zero

15.16.1 Field Documentation

15.16.1.1 is_zero

```
std::map<PolyRef, bool> smtrat::cadcells::datastructures::detail::Assignment↩
Properties::is_zero
```

15.16.1.2 real_roots

```
std::map<PolyRef, carl::ran::real_roots_result<RAN> > smtrat::cadcells↩
::datastructures::detail::AssignmentProperties::real_roots
```

15.17 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

15.17.1 Detailed Description

Represents an Attribute.

15.17.2 Member Typedef Documentation

15.17.2.1 AttributeValue

```
using smtrat::parser::Attribute::AttributeValue = types::AttributeValue
```

15.17.3 Constructor & Destructor Documentation

15.17.3.1 Attribute() [1/4] `smtrat::parser::Attribute::Attribute ( ) [inline]`

15.17.3.2 Attribute() [2/4] `smtrat::parser::Attribute::Attribute ( const std::string & key ) [inline], [explicit]`

15.17.3.3 Attribute() [3/4] `smtrat::parser::Attribute::Attribute ( const std::string & key, const AttributeValue & value ) [inline]`

15.17.3.4 Attribute() [4/4] `smtrat::parser::Attribute::Attribute ( const std::string & key, const boost::optional< AttributeValue > & value ) [inline]`

15.17.4 Member Function Documentation

15.17.4.1 hasValue() `bool smtrat::parser::Attribute::hasValue ( ) const [inline]`

15.17.4.2 simplify() `void smtrat::parser::Attribute::simplify ( ) [inline]`

15.17.5 Field Documentation

15.17.5.1 key `std::string smtrat::parser::Attribute::key`

15.17.5.2 value `AttributeValue smtrat::parser::Attribute::value`

15.18 smtrat::parser::AttributeParser Struct Reference

`#include <Attribute.h>`

Public Member Functions

- AttributeParser ()

Data Fields

- KeywordParser keyword
- AttributeValueParser value
- qi::rule< Iterator, Attribute(), Skipper > main

15.18.1 Constructor & Destructor Documentation

15.18.1.1 AttributeParser() `smtrat::parser::AttributeParser::AttributeParser ( ) [inline]`

15.18.2 Field Documentation

15.18.2.1 keyword `KeywordParser smtrat::parser::AttributeParser::keyword`

15.18.2.2 main `qi::rule<Iterator, Attribute(), Skipper> smtrat::parser::AttributeParser::main`

15.18.2.3 value `AttributeValueParser smtrat::parser::AttributeParser::value`

15.19 smtrat::parser::AttributeValueParser Struct Reference

`#include <Attribute.h>`

Public Types
- typedef conversion::VariantVariantConverter< types::AttributeValue > Converter

Public Member Functions
- AttributeValueParser ()

Data Fields
- SpecConstantParser specconstant
- SymbolParser symbol
- SExpressionParser sexpression
- Converter converter
- qi::rule< Iterator, types::AttributeValue(), Skipper > main

15.19.1 Member Typedef Documentation

15.19.1.1 Converter `typedef conversion::VariantVariantConverter<types::AttributeValue> smtrat::parser::Attrib`

15.19.2 Constructor & Destructor Documentation

15.19.2.1 AttributeValueParser() `smtrat::parser::AttributeValueParser::AttributeValueParser ( ) [inline]`

15.19.3 Field Documentation

15.19.3.1 converter `Converter smtrat::parser::AttributeValueParser::converter`

15.19.3.2 main `qi::rule<Iterator, types::AttributeValue(), Skipper> smtrat::parser::Attribute↩ ValueParser::main`

15.19.3.3 sexpression `SExpressionParser smtrat::parser::AttributeValueParser::sexpression`

15.19.3.4 specconstant `SpecConstantParser smtrat::parser::AttributeValueParser::specconstant`

15.19.3.5 symbol `SymbolParser smtrat::parser::AttributeValueParser::symbol`

15.20 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 monomial↩ Map)

15.20.1 Member Enumeration Documentation

15.20.1.1 AxiomType `enum smtrat::AxiomFactory::AxiomType`

Enumerator

ZERO	
TANGENT_PLANE	
MONOTONICITY	
CONGRUENCE	
ICP	

15.20.2 Member Function Documentation

15.20.2.1 createFormula()

```
FormulasT smtrat::AxiomFactory::createFormula (
    AxiomType axiomType,
    Model linearizedModel,
    MonomialMap monomialMap ) [static]
```

15.21 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.

15.21.1 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 ourside 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.

15.21.2 Constructor & Destructor Documentation

15.21.2.1 Backend()

```
benchmax::Backend::Backend ( ) [inline], [protected]
```

15.21.3 Member Function Documentation

15.21.3.1 addResult()

```
void benchmax::Backend::addResult (
    const Tool * tool,
    const fs::path & file,
    BenchmarkResult && result ) [inline]
```

Add a result.

15.21.3.2 collect_results()

```
virtual bool benchmax::Backend::collect_results (
    const Jobs & ,
    bool ) [inline], [protected], [virtual]
```

15.21.3.3 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.

15.21.3.4 finalize()

```
virtual void benchmax::Backend::finalize ( ) [inline], [protected], [virtual]
```

Hook to allow for asynchronous backends to wait for jobs to terminate.

15.21.3.5 madeProgress()

```
void benchmax::Backend::madeProgress (
    std::size_t files = 1 ) [inline], [protected]
```

Can be called to give information about the current progress, if available.

15.21.3.6 process_results()

```
void benchmax::Backend::process_results (
    const Jobs & jobs,
    bool check_finished ) [inline]
```

15.21.3.7 run()

```
void benchmax::Backend::run (
    const Jobs & jobs,
    bool wait_for_termination ) [inline]
```

Run the list of tools against the list of benchmarks.

15.21.3.8 sanitize_results()

```
void benchmax::Backend::sanitize_results (
    const Jobs & jobs ) const [inline], [protected]
```

15.21.3.9 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.

15.21.3.10 suspendable()

```
bool benchmax::Backend::suspendable ( ) const [inline]
```

15.21.3.11 write_results()

```
void benchmax::Backend::write_results (
    const Jobs & jobs ) const [inline], [protected]
```

15.21.4 Field Documentation

15.21.4.1 mExpectedJobs

```
std::size_t benchmax::Backend::mExpectedJobs [protected]
```

Number of jobs that should be run.

15.21.4.2 mFinishedJobs

```
std::atomic<std::size_t> benchmax::Backend::mFinishedJobs [protected]
```

Number of jobs that are finished.

15.21.4.3 mLastPercent

```
std::atomic<std::size_t> benchmax::Backend::mLastPercent [protected]
```

Percentage of finished jobs when madeProgress() was last called.

15.22 smtrat::Backend< Settings > Class Template Reference

```
#include <Backend.h>
```

Public Member Functions

- Backend ()
- void init (std::vector< carl::Variable > &varOrdering)
- std::size_t dimension ()
- const carl::ran_assignment< Rational > & 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)

15.22.1 Constructor & Destructor Documentation

15.22.1.1 Backend() `template<typename Settings>`

```
smtrat::Backend< Settings >::Backend ( ) [inline]
```

15.22.2 Member Function Documentation

15.22.2.1 addConstraint() [1/2] `template<typename Settings>`

```
void smtrat::Backend< Settings >::addConstraint (
            const ConstraintT & constraint ) [inline]
```

15.22.2.2 addConstraint() [2/2] `template<typename Settings>`

```
void smtrat::Backend< Settings >::addConstraint (
            const size_t level,
            const std::vector< ConstraintT > && constraints ) [inline]
```

15.22.2.3 dimension() `template<typename Settings>`

```
std::size_t smtrat::Backend< Settings >::dimension ( ) [inline]
```

15.22.2.4 getCoveringInformation() `template<typename Settings>`

```
auto& smtrat::Backend< Settings >::getCoveringInformation ( ) [inline]
```

15.22.2.5 getCurrentAssignment() `template<typename Settings>`

```
const carl::ran_assignment<Rational>& smtrat::Backend< Settings >::getCurrentAssignment ( )
[inline]
```

15.22.2.6 getInfeasibleSubset()

```
template<typename Settings>
FormulaSetT smtrat::Backend< Settings >::getInfeasibleSubset ( ) [inline]
```

15.22.2.7 getKnownConstraints() [1/2]

```
template<typename Settings>
auto& smtrat::Backend< Settings >::getKnownConstraints ( ) [inline]
```

15.22.2.8 getKnownConstraints() [2/2]

```
template<typename Settings>
auto& smtrat::Backend< Settings >::getKnownConstraints (
            std::size_t & level ) [inline]
```

15.22.2.9 getUnknownConstraints() [1/2]

```
template<typename Settings>
auto& smtrat::Backend< Settings >::getUnknownConstraints ( ) [inline]
```

15.22.2.10 getUnknownConstraints() [2/2]

```
template<typename Settings>
auto& smtrat::Backend< Settings >::getUnknownConstraints (
            std::size_t & level ) [inline]
```

15.22.2.11 getUnsatCover()

```
template<typename Settings>
Answer smtrat::Backend< Settings >::getUnsatCover (
            const std::size_t level ) [inline]
```

15.22.2.12 init()

```
template<typename Settings>
void smtrat::Backend< Settings >::init (
            std::vector< carl::Variable > & varOrdering ) [inline]
```

15.22.2.13 processUnknownConstraints()

```
template<typename Settings>
void smtrat::Backend< Settings >::processUnknownConstraints (
            const std::size_t & level,
            const bool prune_assignment ) [inline]
```

15.22.2.14 removeConstraint()

```
template<typename Settings>
void smtrat::Backend< Settings >::removeConstraint (
            const ConstraintT & constraint ) [inline]
```

15.22.2.15 resetDerivationToConstraintMap()

```
template<typename Settings>
void smtrat::Backend< Settings >::resetDerivationToConstraintMap ( ) [inline]
```

15.22.2.16 resetStoredData()

```
template<typename Settings>
void smtrat::Backend< Settings >::resetStoredData (
            std::size_t level ) [inline]
```

15.22.2.17 setConstraintsKnown()

```
template<typename Settings>
void smtrat::Backend< Settings >::setConstraintsKnown (
            const std::size_t & level ) [inline]
```

15.22.2.18 setConstraintsUnknown()

```
template<typename Settings>
void smtrat::Backend< Settings >::setConstraintsUnknown (
            const std::size_t & level ) [inline]
```

15.22.2.19 updateAssignment()

```
template<typename Settings>
void smtrat::Backend< Settings >::updateAssignment (
            std::size_t level ) [inline]
```

15.23 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)

15.23.1 Constructor & Destructor Documentation

15.23.1.1 BackendLink()

```
smtrat::BackendLink::BackendLink (
            std::size_t target,
            std::size_t priority,
            const ConditionFunction & cf ) [inline]
```

15.23.2 Member Function Documentation

15.23.2.1 checkCondition()

```
bool smtrat::BackendLink::checkCondition (
            const carl::Condition & c ) const [inline]
```

15.23.2.2 condition()

```
template<typename T >
BackendLink& smtrat::BackendLink::condition (
            const T & f ) [inline]
```

15.23.2.3 getPriority() `std::size_t smtrat::BackendLink::getPriority ( ) const [inline]`

15.23.2.4 getTarget() `std::size_t smtrat::BackendLink::getTarget ( ) const [inline]`

15.23.2.5 id() `BackendLink& smtrat::BackendLink::id ( std::size_t & id ) [inline]`

15.23.2.6 operator<() `bool smtrat::BackendLink::operator< ( const BackendLink & rhs ) const [inline]`

15.23.2.7 priority() `BackendLink& smtrat::BackendLink::priority ( std::size_t p ) [inline]`

15.24 smtrat::BackendSynchronisation Class Reference

`#include <ThreadPool.h>`

Public Member Functions

- BackendSynchronisation ()
- void wait ()
- void notify ()

15.24.1 Constructor & Destructor Documentation

15.24.1.1 BackendSynchronisation() `smtrat::BackendSynchronisation::BackendSynchronisation ( ) [inline]`

15.24.2 Member Function Documentation

15.24.2.1 notify() `void smtrat::BackendSynchronisation::notify ( ) [inline]`

15.24.2.2 wait() `void smtrat::BackendSynchronisation::wait ( ) [inline]`

15.25 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

15.25.1 Member Typedef Documentation

15.25.1.1 History

```
template<typename UnionFind >
using smtrat::Backtrackable< UnionFind >::History = std::vector< std::pair< Timestamp, Union↩
Find > >
```

15.25.1.2 Representative

```
template<typename UnionFind >
using smtrat::Backtrackable< UnionFind >::Representative = Value
```

15.25.1.3 Timestamp

```
template<typename UnionFind >
using smtrat::Backtrackable< UnionFind >::Timestamp = std::pair<Value, Value>
```

15.25.1.4 Value

```
template<typename UnionFind >
using smtrat::Backtrackable< UnionFind >::Value = typename UnionFind::Value
```

15.25.2 Constructor & Destructor Documentation

15.25.2.1 Backtrackable()

```
template<typename UnionFind >
smtrat::Backtrackable< UnionFind >::Backtrackable ( ) [inline]
```

15.25.3 Member Function Documentation

15.25.3.1 backtrack()

```
template<typename UnionFind >
void smtrat::Backtrackable< UnionFind >::backtrack (
            Value const & a,
            Value const & b ) [inline], [noexcept]
```

15.25.3.2 current()

```
template<typename UnionFind >
UnionFind& smtrat::Backtrackable< UnionFind >::current ( ) [inline], [noexcept]
```

15.25.3.3 find()

```
template<typename UnionFind >
auto smtrat::Backtrackable< UnionFind >::find (
            Value const & val ) -> Representative [inline], [noexcept]
```

15.25.3.4 init_size()

```
template<typename UnionFind >
void smtrat::Backtrackable< UnionFind >::init_size (
            size_t size )  [inline], [noexcept]
```

15.25.3.5 merge()

```
template<typename UnionFind >
void smtrat::Backtrackable< UnionFind >::merge (
            Value const & a,
            Value const & b )  [inline], [noexcept]
```

15.25.3.6 origin()

```
template<typename UnionFind >
constexpr auto smtrat::Backtrackable< UnionFind >::origin ( ) const -> Timestamp  [inline],
[constexpr], [noexcept]
```

15.25.3.7 version()

```
template<typename UnionFind >
auto smtrat::Backtrackable< UnionFind >::version (
            Timestamp && ts ) const -> typename History::const_iterator  [inline], [noexcept]
```

15.25.4 Field Documentation

15.25.4.1 history

```
template<typename UnionFind >
History smtrat::Backtrackable< UnionFind >::history
```

15.26 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)
- DerivationRef< Properties > & underlying ()
- 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)

15.26.1 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

Properties	Set of properties (from the operator)

15.26.2 Constructor & Destructor Documentation

15.26.2.1 BaseDerivation()

```
template<typename Properties >
smtrat::cadcells::datastructures::BaseDerivation< Properties >::BaseDerivation (
            Projections & Projections,
            DerivationRef< Properties > underlying,
            size_t level ) [inline]
```

15.26.3 Member Function Documentation

15.26.3.1 contains()

```
template<typename Properties >
template<typename P >
bool smtrat::cadcells::datastructures::BaseDerivation< Properties >::contains (
            const P & property ) const [inline]
```

15.26.3.2 insert()

```
template<typename Properties >
template<typename P >
void smtrat::cadcells::datastructures::BaseDerivation< Properties >::insert (
            P property ) [inline]
```

15.26.3.3 level()

```
template<typename Properties >
size_t smtrat::cadcells::datastructures::BaseDerivation< Properties >::level ( ) const [inline]
```

15.26.3.4 main_var()

```
template<typename Properties >
carl::Variable smtrat::cadcells::datastructures::BaseDerivation< Properties >::main_var ( )
const [inline]
```

15.26.3.5 merge_with()

```
template<typename Properties >
void smtrat::cadcells::datastructures::BaseDerivation< Properties >::merge_with (
            const BaseDerivation< Properties > & other ) [inline]
```

15.26.3.6 polys()

```
template<typename Properties >
PolyPool& smtrat::cadcells::datastructures::BaseDerivation< Properties >::polys ( ) [inline]
```

15.26.3.7 proj()

```
template<typename Properties >
Projections& smtrat::cadcells::datastructures::BaseDerivation< Properties >::proj ( ) [inline]
```

15.26.3.8 properties()

```
template<typename Properties >
template<typename P >
const PropertiesTSet<P>& smtrat::cadcells::datastructures::BaseDerivation< Properties >↩
::properties ( ) const  [inline]
```

15.26.3.9 underlying() [1/2]

```
template<typename Properties >
DerivationRef<Properties>& smtrat::cadcells::datastructures::BaseDerivation< Properties >↩
::underlying ( )  [inline]
```

15.26.3.10 underlying() [2/2]

```
template<typename Properties >
DerivationRef<Properties>& smtrat::cadcells::datastructures::BaseDerivation< Properties >↩
::underlying ( ) const  [inline]
```

15.26.4 Friends And Related Function Documentation

15.26.4.1 merge_underlying

```
template<typename Properties >
template<typename P >
void merge_underlying (
            std::vector< SampledDerivationRef< P >> & derivations )  [friend]
```

15.27 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.

15.27.1 Member Typedef Documentation

15.27.1.1 Constraints

```
template<typename Settings>
using smtrat::cad::BaseProjection< Settings >::Constraints = CADConstraints<Settings::backtracking>
[protected]
```

15.27.2 Constructor & Destructor Documentation

15.27.2.1 BaseProjection()

```
template<typename Settings>
smtrat::cad::BaseProjection< Settings >::BaseProjection (
            const Constraints & c )  [inline], [protected]
```

15.27.3 Member Function Documentation

15.27.3.1 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.

15.27.3.2 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 >.

15.27.3.3 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.

15.27.3.4 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::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Incrementality::NONE
smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, smtrat::cad::Projection< Incrementality::FULL, BT, S
and smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >.

15.27.3.5 callRemoveCallback() template<typename Settings>

```
void smtrat::cad::BaseProjection< Settings >::callRemoveCallback (
            std::size_t level,
            const SampleLiftedWith & slw ) const  [inline], [protected]
```

15.27.3.6 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.

15.27.3.7 dim()

```
template<typename Settings>
std::size_t smtrat::cad::BaseProjection< Settings >::dim ( ) const [inline]
```

Returns the dimension of the projection.

15.27.3.8 empty() [1/2]

```
template<typename Settings>
virtual bool smtrat::cad::BaseProjection< Settings >::empty ( ) [inline], [virtual]
```

15.27.3.9 empty() [2/2]

```
template<typename Settings>
virtual bool smtrat::cad::BaseProjection< Settings >::empty (
            std::size_t level ) const [pure virtual]
```

Implemented in smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, smtrat::cad::Projection< Incremental smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Increme smtrat::qe::cad::Projection< Settings >, smtrat::qe::cad::Projection< smtrat::qe::cad::CADSettings >, and smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >.

15.27.3.10 exportAsDot()

```
template<typename Settings>
virtual void smtrat::cad::BaseProjection< Settings >::exportAsDot (
            std::ostream & ) const [inline], [virtual]
```

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, and smtrat::cad::Projection< Incre

15.27.3.11 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.

15.27.3.12 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.

15.27.3.13 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 >.

15.27.3.14 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.

15.27.3.15 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::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, smtrat::cad::Projection< Incremental smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Increme smtrat::qe::cad::Projection< Settings >, smtrat::qe::cad::Projection< smtrat::qe::cad::CADSettings >, and smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >.

15.27.3.16 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::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, smtrat::cad::Projection< Incremental smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Increme smtrat::qe::cad::Projection< Settings >, smtrat::qe::cad::Projection< smtrat::qe::cad::CADSettings >, and smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >.

15.27.3.17 isPurged() template<typename Settings>

```
bool smtrat::cad::BaseProjection< Settings >::isPurged (
            std::size_t level,
            std::size_t id )  [inline], [protected]
```

15.27.3.18 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.

15.27.3.19 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::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Incrementality::FULL smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >, and smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >.

15.27.3.20 reset() template<typename Settings>

```
void smtrat::cad::BaseProjection< Settings >::reset ( ) [inline]
```

Resets all datastructures, use the given variables from now on.

15.27.3.21 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.

15.27.3.22 size() [1/2] template<typename Settings>

```
std::size_t smtrat::cad::BaseProjection< Settings >::size ( ) const [inline]
```

15.27.3.23 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::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, smtrat::cad::Projection< Incremental smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Increme smtrat::qe::cad::Projection< Settings >, smtrat::qe::cad::Projection< smtrat::qe::cad::CADSettings >, and smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >.

15.27.3.24 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.

15.27.3.25 vars() template<typename Settings>

```
const auto& smtrat::cad::BaseProjection< Settings >::vars ( ) const [inline]
```

Returns the variables used for projection.

15.27.4 Field Documentation

15.27.4.1 mConstraints

```
template<typename Settings>
const Constraints& smtrat::cad::BaseProjection< Settings >::mConstraints [protected]
```

15.27.4.2 mInfo

```
template<typename Settings>
ProjectionInformation smtrat::cad::BaseProjection< Settings >::mInfo [protected]
```

Additional info on projection, projection levels and projection polynomials.

15.27.4.3 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.

15.27.4.4 mOperator

```
template<typename Settings>
ProjectionOperator smtrat::cad::BaseProjection< Settings >::mOperator [protected]
```

The projection operator.

15.27.4.5 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.

15.28 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

15.28.1 Field Documentation

15.28.1.1 backtracking

```
constexpr Backtracking smtrat::cad::BaseSettings::backtracking = Backtracking::ORDERED
[static], [constexpr]
```

15.28.1.2 coreHeuristic

`constexpr CoreHeuristic smtrat::cad::BaseSettings::coreHeuristic = cad::CoreHeuristic::PreferProjection [static], [constexpr]`

15.28.1.3 fullSampleComparator

`constexpr FullSampleCompareStrategy smtrat::cad::BaseSettings↩::fullSampleComparator = cad::FullSampleCompareStrategy::Default [static], [constexpr]`

15.28.1.4 incrementality

`constexpr Incrementality smtrat::cad::BaseSettings::incrementality = Incrementality::NONE [static], [constexpr]`

15.28.1.5 misHeuristic

`constexpr MISHeuristic smtrat::cad::BaseSettings::misHeuristic = cad::MISHeuristic::GRE [static], [constexpr]`

15.28.1.6 projectionComparator

`constexpr ProjectionCompareStrategy smtrat::cad::BaseSettings↩::projectionComparator = cad::ProjectionCompareStrategy::Default [static], [constexpr]`

15.28.1.7 projectionOperator

`constexpr ProjectionType smtrat::cad::BaseSettings::projection↩Operator = cad::ProjectionType::McCallum [static], [constexpr]`

15.28.1.8 sampleComparator

`constexpr SampleCompareStrategy smtrat::cad::BaseSettings::sample↩Comparator = cad::SampleCompareStrategy::Default [static], [constexpr]`

15.28.1.9 simplifyProjectionByBounds

`constexpr bool smtrat::cad::BaseSettings::simplify↩ProjectionByBounds = true [static], [constexpr]`

15.28.1.10 trivialSampleRadius

`constexpr std::size_t smtrat::cad::BaseSettings::trivialSample↩Radius = 1 [static], [constexpr]`

15.29 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

15.29.1 Member Function Documentation

15.29.1.1 operator()() [1/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const BinaryExpression & expr ) const [inline]
```

15.29.1.2 operator()() [2/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const carl::Variable & expr ) const [inline]
```

15.29.1.3 operator()() [3/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const ExpressionContent * _ec ) const [inline]
```

15.29.1.4 operator()() [4/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const ITEExpression & expr ) const [inline]
```

15.29.1.5 operator()() [5/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const NaryExpression & expr ) const [inline]
```

15.29.1.6 operator()() [6/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const QuantifierExpression & expr ) const [inline]
```

15.29.1.7 operator()() [7/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const UnaryExpression & expr ) const [inline]
```

15.29.1.8 simplify() [1/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const BinaryExpression & ) const [inline], [protected], [virtual]
```

15.29.1.9 simplify() [2/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::↩
BaseSimplifier::simplify (
            const carl::Variable &  ) const  [inline], [protected], [virtual]
```

15.29.1.10 simplify() [3/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::↩
BaseSimplifier::simplify (
            const ITEExpression &  ) const  [inline], [protected], [virtual]
```

15.29.1.11 simplify() [4/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::↩
BaseSimplifier::simplify (
            const NaryExpression &  ) const  [inline], [protected], [virtual]
```

Reimplemented in smtrat::expression::simplifier::DuplicateSimplifier, smtrat::expression::simplifier::MergeSimplifier, and smtrat::expression::simplifier::SingletonSimplifier.

15.29.1.12 simplify() [5/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::↩
BaseSimplifier::simplify (
            const QuantifierExpression &  ) const  [inline], [protected], [virtual]
```

15.29.1.13 simplify() [6/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::↩
BaseSimplifier::simplify (
            const UnaryExpression &  ) const  [inline], [protected], [virtual]
```

Reimplemented in smtrat::expression::simplifier::NegationSimplifier.

15.30 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)

15.30.1 Constructor & Destructor Documentation

15.30.1.1 BaseType() [1/2]

```
smtrat::cad::Origin::BaseType::BaseType (
    std::size_t level,
    std::size_t id ) [inline], [explicit]
```

15.30.1.2 BaseType() [2/2]

```
smtrat::cad::Origin::BaseType::BaseType (
    std::size_t lvl,
    std::size_t id1,
    std::size_t id2 ) [inline]
```

15.30.2 Member Function Documentation

15.30.2.1 activate()

```
void smtrat::cad::Origin::BaseType::activate (
    const carl::Bitset & ids ) [inline]
```

15.30.2.2 active()

```
bool smtrat::cad::Origin::BaseType::active ( ) const [inline]
```

15.30.2.3 deactivate()

```
void smtrat::cad::Origin::BaseType::deactivate (
    const carl::Bitset & ids ) [inline]
```

15.30.2.4 operator<()

```
bool smtrat::cad::Origin::BaseType::operator< (
    const BaseType & bt ) const [inline]
```

15.30.2.5 operator==()

```
bool smtrat::cad::Origin::BaseType::operator== (
    const BaseType & bt ) const [inline]
```

15.30.3 Friends And Related Function Documentation

15.30.3.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const BaseType & bt ) [friend]
```

15.30.4 Field Documentation

15.30.4.1 ec_active

```
bool smtrat::cad::Origin::BaseType::ec_active = true
```

15.30.4.2 first

```
std::size_t smtrat::cad::Origin::BaseType::first
```

15.30.4.3 first_active

```
bool smtrat::cad::Origin::BaseType::first_active = true
```

15.30.4.4 level

```
std::size_t smtrat::cad::Origin::BaseType::level
```

15.30.4.5 second `std::size_t smtrat::cad::Origin::BaseType::second`

15.30.4.6 second_active `bool smtrat::cad::Origin::BaseType::second_active = true`

15.31 smtrat::BEModule< Settings > Class Template Reference

`#include <BEModule.h>`

Public Types

- typedef Settings SettingsType
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::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 smtrat::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 _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.

- 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 &_branching↩Value) 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 _preferLeft↩Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩::vector< FormulaT >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩CaseWeak=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.

15.31.1 Member Typedef Documentation

15.31.1.1 SettingsType `template<typename Settings >`

```
typedef Settings smtrat::BEModule< Settings >::SettingsType
```

15.31.2 Member Enumeration Documentation

15.31.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

15.31.3 Constructor & Destructor Documentation

15.31.3.1 BEModule() `template<class Settings >`

```
smtrat::BEModule< Settings >::BEModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = NULL )
```

15.31.3.2 ~BEModule() `template<class Settings >`

```
smtrat::BEModule< Settings >::~BEModule ( )
```

15.31.4 Member Function Documentation

15.31.4.1 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

_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.

15.31.4.2 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 >.

15.31.4.3 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

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 smtrat::SATModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::VSModule< Settings >, smtrat::PFEModule< Settings >, smtrat::EQModule< Setting smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Sett smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, and smtrat::GBModule< Settings >.

15.31.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.31.4.5 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.

15.31.4.6 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.

15.31.4.7 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.

15.31.4.8 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.

15.31.4.9 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.

15.31.4.10 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

_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.

15.31.4.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const  [inline],
[inherited]
```

Returns All satisfying assignments, if existent.

15.31.4.12 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.

15.31.4.13 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.

15.31.4.14 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.

15.31.4.15 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.

15.31.4.16 branchAt() [1/6]

```
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]
```

15.31.4.17 branchAt() [2/6]

```
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]
```

15.31.4.18 branchAt() [3/6]

```
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]
```

15.31.4.19 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.31.4.20 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
    const typename P::PolyType & _poly,
    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]
```

15.31.4.21 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false )  [inline], [protected], [inherited]
```

15.31.4.22 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

15.31.4.23 checkCore()

```
template<class 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.

15.31.4.24 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
            std::vector< FormulaSetT >::const_iterator _infsubset,
            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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.31.4.25 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.

15.31.4.26 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.31.4.27 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.31.4.28 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.31.4.29 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.31.4.30 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.31.4.31 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

15.31.4.32 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.

15.31.4.33 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.31.4.34 constraintsToInform() `const carl::FastSet<FormulaT>& smtrat::Module::constraintsTo↩ Inform ( ) const [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.31.4.35 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

15.31.4.36 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)

15.31.4.37 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

_constraint	The constraint to remove from internal data structures.

15.31.4.38 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.31.4.39 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.31.4.40 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.31.4.41 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceived↩
VariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.31.4.42 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.31.4.43 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.

15.31.4.44 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.

15.31.4.45 freeSplittingVariable()

```
static void smtrat::Module::freeSplittingVariable (
    const FormulaT & _splittingVariable ) [inline], [static], [inherited]
```

15.31.4.46 generateTrivialInfeasibleSubset()

```
void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]
```

Stores the trivial infeasible subset being the set of received formulas.

15.31.4.47 getBackendsAllModels()

```
void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]
```

Stores all models of a backend in the list of all models of this module.

15.31.4.48 getBackendsModel()

```
void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]
```

15.31.4.49 getInfeasibleSubsets() [1/2]

```
void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]
```

Copies the infeasible subsets of the passed formula.

15.31.4.50 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.31.4.51 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.31.4.52 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.

15.31.4.53 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.31.4.54 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.31.4.55 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.31.4.56 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 reveived formula, if the Boolean is true.

Reimplemented from smtrat::Module.

15.31.4.57 hasLemmas()

`bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

15.31.4.58 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).

15.31.4.59 id()

`std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns A unique ID to identify this module instance.

15.31.4.60 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.

15.31.4.61 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.31.4.62 informBackends()

```
void smtrat::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.

15.31.4.63 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, and smtrat::ICPModule< ICPSettings1 >.

15.31.4.64 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.31.4.65 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::FPPModule< Settings >.

15.31.4.66 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.31.4.67 isLemmaLevel() `bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.31.4.68 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().

15.31.4.69 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.

15.31.4.70 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.31.4.71 model()

```
const Model& smtrat::Module::model ( ) const [inline], [inherited]
```

Returns The assignment of the current satisfiable result, if existent.

15.31.4.72 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.

15.31.4.73 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.

15.31.4.74 objective()

```
carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]
```

15.31.4.75 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
    const FormulaT & _origin ) const [protected], [inherited]
```

15.31.4.76 passedFormulaBegin() `ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula. TODO: disable this method

15.31.4.77 passedFormulaEnd() `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula.

15.31.4.78 pPassedFormula() `const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to the backends of this module.

15.31.4.79 pReceivedFormula() `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to this module.

15.31.4.80 print() `void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.31.4.81 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

_out	The stream to print the assignment on.

15.31.4.82 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.31.4.83 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

_out	The stream to print the assignment on.

15.31.4.84 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.31.4.85 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.31.4.86 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.

15.31.4.87 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

15.31.4.88 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAsInfeasible↩Subset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

15.31.4.89 receivedVariable()

`bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

15.31.4.90 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

_subformula	The sub formula of the received formula to remove.

Reimplemented from smtrat::Module.

15.31.4.91 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

formula	The sub formula of the received formula to remove.

Reimplemented in smtrat::SATModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::VSModule< Settings >, smtrat::PFEModule< Settings >, smtrat::EQModule< Setting smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Sett smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, and smtrat::GBModule< Settings >.

15.31.4.92 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.31.4.93 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins
(
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

15.31.4.94 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline],
[inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.31.4.95 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.31.4.96 runBackends() [1/2]

```
virtual Answer smtrat::PModule::runBackends ( ) [inline],
[virtual], [inherited]
```

Reimplemented from smtrat::Module.

15.31.4.97 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.

15.31.4.98 setId()

```
void smtrat::Module::setId (
    std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.31.4.99 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.

15.31.4.100 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.

15.31.4.101 splitUnequalConstraint()

```
void smtrat::Module::splitUnequalConstraint (
    const FormulaT & _unequalConstraint ) [protected], [inherited]
```

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.31.4.102 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.31.4.103 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 >.

15.31.4.104 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.31.4.105 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.

15.31.4.106 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).

15.31.5 Field Documentation

15.31.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.31.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.31.5.3 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.

15.31.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.31.5.5 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

15.31.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.31.5.7 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.

15.31.5.8 mFirstUncheckedReceivedSubformula `ModuleInput::const_iterator smtrat::Module::m↩ FirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

15.31.5.9 mFoundAnswer `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.31.5.10 mFullCheck `bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.31.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.31.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.31.5.13 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.

15.31.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.31.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.31.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.31.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.31.5.18 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.

15.31.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]`

Reusable splitting variables.

15.31.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.31.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.31.5.22 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.

15.31.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

15.31.5.24 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).

15.31.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.32 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.

15.32.1 Detailed Description

Results for a single benchmark run.

15.32.2 Member Function Documentation

15.32.2.1 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.

15.32.2.2 forget()

```
void benchmax::BenchmarkResult::forget ( ) const  [inline]
```

15.32.2.3 get_path()

```
auto benchmax::BenchmarkResult::get_path ( ) const  [inline]
```

15.32.2.4 restore()

```
void benchmax::BenchmarkResult::restore ( ) const  [inline]
```

15.32.2.5 store() [1/2]

```
void benchmax::BenchmarkResult::store ( ) const  [inline]
```

15.32.2.6 store() [2/2]

```
void benchmax::BenchmarkResult::store (
            size_t id ) const  [inline]
```

15.32.3 Field Documentation

15.32.3.1 additional

```
std::map<std::string, std::string> benchmax::BenchmarkResult::additional
[mutable]
```

Arbitrary additional information that can be provided by the tool class.

15.32.3.2 answer `std::string benchmax::BenchmarkResult::answer`

Answer string.

15.32.3.3 exitCode `int benchmax::BenchmarkResult::exitCode`

Shell exit code.

15.32.3.4 peak_memory_kbytes `std::size_t benchmax::BenchmarkResult::peak_memory_kbytes`

Peak memory usage.

15.32.3.5 stderr `std::string benchmax::BenchmarkResult::stderr`

Error output (mostly for parsing the answer and additional information).

15.32.3.6 stdout `std::string benchmax::BenchmarkResult::stdout`

Standard output (mostly for parsing the answer and additional information).

15.32.3.7 stored_id `size_t benchmax::BenchmarkResult::stored_id = 0 [mutable]`

Identifier for temporary file.

15.32.3.8 time `std::chrono::milliseconds benchmax::BenchmarkResult::time`

Runtime in milliseconds.

15.33 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.

15.33.1 Detailed Description

A set of benchmarks from some common base directory.

15.33.2 Member Function Documentation

15.33.2.1 add_directory() `void benchmax::BenchmarkSet::add_directory ( const std::filesystem::path & dir )`

Recursively find all benchmarks from this directory.

15.33.2.2 begin() `auto benchmax::BenchmarkSet::begin ( ) const [inline]`

Begin iterator.

15.33.2.3 end() `auto benchmax::BenchmarkSet::end ( ) const [inline]`

End iterator.

15.33.2.4 size() `std::size_t benchmax::BenchmarkSet::size ( ) const [inline]`

Number of files.

15.34 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.

15.34.1 Detailed Description

Settings for benchmarks.

15.34.2 Field Documentation

15.34.2.1 grace_time `carl::settings::duration benchmax::settings::BenchmarkSettings::grace_time`

Grace time in seconds.

15.34.2.2 input_directories `std::vector<std::filesystem::path> benchmax::settings::BenchmarkSettings::input_directories`

Lift of input directories.

15.34.2.3 input_directories_common_prefix `std::filesystem::path benchmax::settings::BenchmarkSettings::input_directories_common_prefix`

Common prefix of input directories (to shorten filenames in output).

15.34.2.4 limit_memory `carl::settings::binary_quantity benchmax::settings::BenchmarkSettings::limit_memory`

Memory limit in megabytes.

15.34.2.5 limit_time `carl::settings::duration benchmax::settings::BenchmarkSettings::limit_time`

Time limit in seconds.

15.34.2.6 output_dir `std::filesystem::path benchmax::settings::BenchmarkSettings::output_dir`

Output directory.

15.34.2.7 output_file_xml `std::filesystem::path benchmax::settings::BenchmarkSettings::output_file_xml`

Filename of xml file.

15.35 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)

15.35.1 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.

15.35.2 Member Typedef Documentation

15.35.2.1 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`

15.35.2.2 Data

`template<class Class, typename FirstKeyType, FirstKeyType Class::* FirstKeyName, typename SecondKeyType, SecondKeyType Class::* SecondKeyName>`
`typedef std::forward_list<Class> smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >::Data`

15.35.2.3 Iterator

`template<class Class, typename FirstKeyType, FirstKeyType Class::* FirstKeyName, typename SecondKeyType, SecondKeyType Class::* SecondKeyName>`
`typedef Data::iterator smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >::Iterator`

15.35.3 Member Function Documentation

15.35.3.1 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]`

15.35.3.2 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]`

15.35.3.3 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]`

15.35.3.4 end() [1/2]

```
template<class Class, typename FirstKeyType, FirstKeyType Class::* First↩
KeyName, typename SecondKeyType, SecondKeyType Class::* SecondKeyName>
ConstIterator smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName
>::end ( ) const  [inline], [noexcept]
```

15.35.3.5 end() [2/2]

```
template<class Class, typename FirstKeyType, FirstKeyType Class::* First↩
KeyName, typename SecondKeyType, SecondKeyType Class::* SecondKeyName>
Iterator smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >::end
( )  [inline], [noexcept]
```

15.35.3.6 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]
```

15.35.3.7 firstFind()

```
template<class Class, typename FirstKeyType, FirstKeyType Class::* First↩
KeyName, typename SecondKeyType, SecondKeyType Class::* SecondKeyName>
Iterator smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >↩
::firstFind (
            const FirstKeyType & firstKey )  [inline]
```

15.35.3.8 secondAt()

```
template<class Class, typename FirstKeyType, FirstKeyType Class::* First↩
KeyName, typename SecondKeyType, SecondKeyType Class::* SecondKeyName>
Class& smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >↩
::secondAt (
            const SecondKeyType & secondKey )  [inline]
```

15.35.3.9 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]
```

15.36 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

15.36.1 Constructor & Destructor Documentation

15.36.1.1 BinaryExpression()

```
smtrat::expression::BinaryExpression::BinaryExpression (
    BinaryType _type,
    Expression && _lhs,
    Expression && _rhs ) [inline]
```

15.36.2 Field Documentation

15.36.2.1 lhs

```
Expression smtrat::expression::BinaryExpression::lhs
```

15.36.2.2 rhs

```
Expression smtrat::expression::BinaryExpression::rhs
```

15.36.2.3 type

```
BinaryType smtrat::expression::BinaryExpression::type
```

15.37 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 isEmpty () const
- std::size_t size () const
- void insert (const T &item)
- T extractMin ()
- void clear ()

15.37.1 Constructor & Destructor Documentation

15.37.1.1 BinaryHeap() [1/2]

```
template<typename T >
smtrat::datastructures::BinaryHeap< T >::BinaryHeap (
    std::size_t initialSize = 100 ) [inline], [explicit]
```

15.37.1.2 BinaryHeap() [2/2]

```
template<typename T >
smtrat::datastructures::BinaryHeap< T >::BinaryHeap (
    const std::vector< T > & elements ) [inline], [explicit]
```

15.37.2 Member Function Documentation

15.37.2.1 clear()

```
template<typename T >
void smtrat::datastructures::BinaryHeap< T >::clear ( ) [inline]
```

15.37.2.2 extractMin()

```
template<typename T >
T smtrat::datastructures::BinaryHeap< T >::extractMin ( ) [inline]
```

15.37.2.3 insert()

```
template<typename T >
void smtrat::datastructures::BinaryHeap< T >::insert (
    const T & item ) [inline]
```

15.37.2.4 isEmpty() `template<typename T >`

`bool smtrat::datastructures::BinaryHeap< T >::isEmpty ( ) const [inline]`

15.37.2.5 size() `template<typename T >`

`std::size_t smtrat::datastructures::BinaryHeap< T >::size ( ) const [inline]`

15.38 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

15.38.1 Detailed Description

Parses binaries: `#b[01]+`

15.38.2 Member Typedef Documentation

15.38.2.1 ITRange `typedef boost::iterator_range<Iterator> smtrat::parser::BinaryParser::ITRange`

15.38.3 Constructor & Destructor Documentation

15.38.3.1 BinaryParser() `smtrat::parser::BinaryParser::BinaryParser ( ) [inline]`

15.38.4 Member Function Documentation

15.38.4.1 build()

```
FixedWidthConstant<Integer> smtrat::parser::BinaryParser::build (
    const ITRange & itr,
    const Integer & val ) [inline]
```

15.38.5 Field Documentation

15.38.5.1 main `qi::rule<Iterator, FixedWidthConstant<Integer>), Skipper, qi::locals<Integer> > smtrat::parser::BinaryParser::main`

15.38.5.2 main2 `qi::rule<Iterator, FixedWidthConstant<Integer>), Skipper> smtrat::parser::↩ BinaryParser::main2`

15.38.5.3 number `qi::uint_parser<Integer,2,1,-1> smtrat::parser::BinaryParser::number`

15.39 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

15.39.1 Detailed Description

Implements the theory of bitvectors.

15.39.2 Member Typedef Documentation

15.39.2.1 OperatorType

```
using smtrat::parser::BitvectorTheory::OperatorType = carl::BVTermType
```

15.39.3 Constructor & Destructor Documentation

15.39.3.1 BitvectorTheory()

```
smtrat::parser::BitvectorTheory::BitvectorTheory (
    ParserState * state )
```

15.39.4 Member Function Documentation

15.39.4.1 addConstants()

```
static void smtrat::parser::AbstractTheory::addConstants (
    qi::symbols< char, types::ConstType > & ) [inline], [static], [inherited]
```

Initialize the global symbol table for constants.

15.39.4.2 addSimpleSorts()

```
void smtrat::parser::BitvectorTheory::addSimpleSorts (
    qi::symbols< char, carl::Sort > & sorts ) [static]
```

15.39.4.3 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.

15.39.4.4 expandDistinct()

```
template<typename T , typename Builder >
FormulaT smtrat::parser::AbstractTheory::expandDistinct (
    const std::vector< T > & values,
    const Builder & neqBuilder ) [inline], [inherited]
```

15.39.4.5 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.

15.39.4.6 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.

15.39.4.7 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.

15.39.4.8 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.

15.39.4.9 refreshVariable()

```
bool smtrat::parser::BitvectorTheory::refreshVariable (
    const types::VariableType & var,
    types::VariableType & subject,
    TheoryError & errors )  [virtual]
```

Reimplemented from smtrat::parser::AbstractTheory.

15.39.4.10 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.

15.39.5 Field Documentation

15.39.5.1 bvSort

```
carl::Sort smtrat::parser::BitvectorTheory::bvSort
```

15.39.5.2 state

```
ParserState* smtrat::parser::AbstractTheory::state  [inherited]
```

15.39.5.3 termConverter

```
conversion::VariantConverter<types::BVTerm> smtrat::parser::Bitvector↩
Theory::termConverter
```

15.39.5.4 vectorConverter `conversion::VectorVariantConverter<types::BVTerm> smtrat::parser::↩ BitvectorTheory::vectorConverter`

15.40 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

15.40.1 Detailed Description

Types of the theory of bitvectors.

15.40.2 Member Typedef Documentation

15.40.2.1 ConstTypes `typedef mpl::vector<BVVariable, FixedWidthConstant<Integer>, BVTerm> smtrat::parser::types::BitvectorTheory::ConstTypes`

15.40.2.2 ExpressionTypes `typedef mpl::vector<BVVariable, FixedWidthConstant<Integer>, BVTerm, BVConstraint> smtrat::parser::types::BitvectorTheory::ExpressionTypes`

15.40.2.3 TermType `typedef carl::mpl_variant_of<TermTypes>::type smtrat::parser::types::BitvectorTheory::Term`

15.40.2.4 TermTypes `typedef mpl::vector<BVVariable, FixedWidthConstant<Integer>, BVTerm, BVConstraint> smtrat::parser::types::BitvectorTheory::TermTypes`

15.40.2.5 VariableTypes `typedef mpl::vector<BVVariable> smtrat::parser::types::BitvectorTheory::VariableTypes`

15.41 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)

15.41.1 Constructor & Destructor Documentation

15.41.1.1 BlastedConstr() [1/4] `smtrat::BlastedConstr::BlastedConstr ( ) [inline]`

15.41.1.2 BlastedConstr() [2/4]

```
smtrat::BlastedConstr::BlastedConstr (
    const FormulaT & _formula,
    const FormulasT & _constraints ) [inline]
```

15.41.1.3 BlastedConstr() [3/4]

```
smtrat::BlastedConstr::BlastedConstr (
    const FormulaT & _formula ) [inline]
```

15.41.1.4 BlastedConstr() [4/4]

```
smtrat::BlastedConstr::BlastedConstr (
    bool _satisfied ) [inline]
```

15.41.2 Member Function Documentation

15.41.2.1 constraints()

```
const FormulasT& smtrat::BlastedConstr::constraints ( ) const [inline]
```

15.41.2.2 formula()

```
const FormulaT& smtrat::BlastedConstr::formula ( ) const [inline]
```

15.41.3 Friends And Related Function Documentation

15.41.3.1 operator<<

```
std::ostream& operator<< (
    std::ostream & _out,
    const BlastedConstr & _constr ) [friend]
```

15.42 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 isConstant () const
- const Integer & constant () const
- const AnnotatedBVTerm & term () const
- const FormulasT & constraints () const
- const Integer & lowerBound () const
- const Integer & upperBound () const

Friends

- std::ostream & operator<< (std::ostream &_out, const BlastedPoly &_poly)

15.42.1 Constructor & Destructor Documentation

15.42.1.1 BlastedPoly() [1/5] `smtrat::BlastedPoly::BlastedPoly ( ) [inline]`

15.42.1.2 BlastedPoly() [2/5]

```
smtrat::BlastedPoly::BlastedPoly (
    Integer _constant ) [inline]
```

15.42.1.3 BlastedPoly() [3/5]

```
smtrat::BlastedPoly::BlastedPoly (
    Integer _constant,
    FormulasT _constraints ) [inline]
```

15.42.1.4 BlastedPoly() [4/5]

```
smtrat::BlastedPoly::BlastedPoly (
    AnnotatedBVTerm _term ) [inline]
```

15.42.1.5 BlastedPoly() [5/5]

```
smtrat::BlastedPoly::BlastedPoly (
    AnnotatedBVTerm _term,
    FormulasT _constraints ) [inline]
```

15.42.2 Member Function Documentation

15.42.2.1 constant()

```
const Integer& smtrat::BlastedPoly::constant ( ) const [inline]
```

15.42.2.2 constraints()

```
const FormulasT& smtrat::BlastedPoly::constraints ( ) const [inline]
```

15.42.2.3 isConstant()

```
bool smtrat::BlastedPoly::isConstant ( ) const [inline]
```

15.42.2.4 lowerBound()

```
const Integer& smtrat::BlastedPoly::lowerBound ( ) const [inline]
```

15.42.2.5 term()

```
const AnnotatedBVTerm& smtrat::BlastedPoly::term ( ) const [inline]
```

15.42.2.6 upperBound()

```
const Integer& smtrat::BlastedPoly::upperBound ( ) const [inline]
```

15.42.3 Friends And Related Function Documentation

15.42.3.1 operator<<

```
std::ostream& operator<< (
    std::ostream & _out,
    const BlastedPoly & _poly ) [friend]
```

15.43 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)

15.43.1 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".

15.43.2 Member Function Documentation

15.43.2.1 assignedVariables()

`const auto& smtrat::mcsat::Bookkeeping::assignedVariables ( ) const [inline]`

15.43.2.2 assignments()

`const auto& smtrat::mcsat::Bookkeeping::assignments ( ) const [inline]`

15.43.2.3 constraints()

`const auto& smtrat::mcsat::Bookkeeping::constraints ( ) const [inline]`

15.43.2.4 model()

`const auto& smtrat::mcsat::Bookkeeping::model ( ) const [inline]`

15.43.2.5 mvBounds()

`const auto& smtrat::mcsat::Bookkeeping::mvBounds ( ) const [inline]`

15.43.2.6 popAssignment()

```
void smtrat::mcsat::Bookkeeping::popAssignment (
    carl::Variable v ) [inline]
```

15.43.2.7 popConstraint()

```
void smtrat::mcsat::Bookkeeping::popConstraint (
    const FormulaT & f ) [inline]
```

15.43.2.8 pushAssignment()

```
void smtrat::mcsat::Bookkeeping::pushAssignment (
    carl::Variable v,
    const ModelValue & mv,
    const FormulaT & f ) [inline]
```

15.43.2.9 pushConstraint()

```
void smtrat::mcsat::Bookkeeping::pushConstraint (
    const FormulaT & f ) [inline]
```

Assert a constraint/literal.

15.43.2.10 updateVariables()

```
void smtrat::mcsat::Bookkeeping::updateVariables (
    const carl::Variables & variables ) [inline]
```

15.43.2.11 variables()

```
const auto& smtrat::mcsat::Bookkeeping::variables ( ) const [inline]
```

15.44 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

15.44.1 Detailed Description

Implements the theory of bitvectors.

15.44.2 Constructor & Destructor Documentation

15.44.2.1 BooleanEncodingTheory()

```
smtrat::parser::BooleanEncodingTheory::BooleanEncodingTheory (
    ParserState * state )
```

15.44.3 Member Function Documentation

15.44.3.1 addConstants()

```
static void smtrat::parser::AbstractTheory::addConstants (
    qi::symbols< char, types::ConstType > & ) [inline], [static], [inherited]
```

Initialize the global symbol table for constants.

15.44.3.2 addSimpleSorts()

```
static void smtrat::parser::AbstractTheory::addSimpleSorts (
    qi::symbols< char, carl::Sort > & ) [inline], [static], [inherited]
```

Initialize the global symbol table for simple sorts.

15.44.3.3 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::ArithmeticTheory, smtrat::parser::CoreTheory, and smtrat::parser::BitvectorTheory.

15.44.3.4 expandDistinct()

```
template<typename T , typename Builder >
FormulaT smtrat::parser::AbstractTheory::expandDistinct (
    const std::vector< T > & values,
    const Builder & neqBuilder ) [inline], [inherited]
```

15.44.3.5 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::BitvectorTheory, smtrat::parser::ArithmeticTheory, and smtrat::parser::CoreTheory.

15.44.3.6 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::ArithmeticTheory, smtrat::parser::BitvectorTheory, smtrat::parser::CoreTheory, and smtrat::parser::UninterpretedTheory.

15.44.3.7 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::ArithmeticTheory, smtrat::parser::BitvectorTheory, and smtrat::parser::CoreTheory.

15.44.3.8 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.

15.44.3.9 refreshVariable()

```
virtual bool smtrat::parser::AbstractTheory::refreshVariable (
    const types::VariableType & ,
    types::VariableType & ,
    TheoryError & errors )  [inline], [virtual], [inherited]
```

Reimplemented in smtrat::parser::BitvectorTheory.

15.44.3.10 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.

15.44.4 Field Documentation

15.44.4.1 state

```
ParserState* smtrat::parser::AbstractTheory::state  [inherited]
```

15.45 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

15.45.1 Constructor & Destructor Documentation

15.45.1.1 BoolOption()

```
Minisat::BoolOption::BoolOption (
    const char * c,
    const char * n,
    const char * d,
    bool v ) [inline]
```

15.45.2 Member Function Documentation

15.45.2.1 getHelpPrefixString()

```
static const char*& Minisat::Option::getHelpPrefixString ( ) [inline], [static], [protected], [inherited]
```

15.45.2.2 getOptionList()

```
static vec<Option*>& Minisat::Option::getOptionList ( ) [inline], [static], [protected], [inherited]
```

15.45.2.3 getUsageString()

```
static const char*& Minisat::Option::getUsageString ( ) [inline], [static], [protected], [inherited]
```

15.45.2.4 help()

```
virtual void Minisat::BoolOption::help (
    bool verbose = false ) [inline], [virtual]
```

Implements Minisat::Option.

15.45.2.5 operator bool()

```
Minisat::BoolOption::operator bool (
    void ) const [inline]
```

15.45.2.6 operator=()

```
BoolOption& Minisat::BoolOption::operator= (
    bool b ) [inline]
```

15.45.3 Field Documentation

15.45.3.1 category

```
const char* Minisat::Option::category [protected], [inherited]
```

15.45.3.2 description `const char* Minisat::Option::description [protected], [inherited]`

15.45.3.3 name `const char* Minisat::Option::name [protected], [inherited]`

15.45.3.4 type_name `const char* Minisat::Option::type_name [protected], [inherited]`

15.46 smtrat::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)

15.46.1 Constructor & Destructor Documentation

15.46.1.1 BoolUEQRewriter()

```
smtrat::BoolUEQRewriter::BoolUEQRewriter (
    CollectBoolsInUEQs && collected ) [inline]
```

15.46.2 Member Function Documentation

15.46.2.1 addDomainConstraints()

```
void smtrat::BoolUEQRewriter::addDomainConstraints (
    std::unordered_multimap< FormulaT, FormulaT > & formulaMap ) [inline]
```

15.46.2.2 addFormulas()

```
void smtrat::BoolUEQRewriter::addFormulas (
    std::unordered_multimap< FormulaT, FormulaT > & formulaMap ) [inline]
```

15.46.2.3 rewrite_bool()

```
FormulaT smtrat::BoolUEQRewriter::rewrite_bool (
    const FormulaT & formula ) [inline]
```

15.46.2.4 setCollected()

```
void smtrat::BoolUEQRewriter::setCollected (
    CollectBoolsInUEQs && collected ) [inline]
```

15.47 smtrat::lra::Bound< T1, T2 > Class Template Reference

Stores a bound, which could be an upper "<= b" or a lower bound ">= 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 = ">=", UPPER = "<=", 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)

15.47.1 Detailed Description

template<typename T1, typename T2>
class smtrat::lra::Bound< T1, T2 >

Stores a bound, which could be an upper "<= b" or a lower bound ">= b" for a bound value b.

In the case that "<= b" and ">= b" hold this bound is a so called equal bound represented by "==b".

15.47.2 Member Typedef Documentation

15.47.2.1 BoundSet

```
template<typename T1, typename T2>
typedef carl::PointerSet<Bound<T1, T2> > smtrat::lra::Bound< T1, T2 >::BoundSet
```

A set of pointer to bounds.

15.47.3 Member Enumeration Documentation

15.47.3.1 Type

```
template<typename T1, typename T2>
enum smtrat::lra::Bound::Type
```

The type of the bound: LOWER = ">=", UPPER = "<=", EQUAL "==".

Enumerator

LOWER	
UPPER	
EQUAL	

15.47.4 Constructor & Destructor Documentation

15.47.4.1 Bound() [1/2]

```
template<typename T1, typename T2>
smtrat::lra::Bound< T1, T2 >::Bound ( ) [delete]
```

15.47.4.2 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	

15.47.4.3 ~Bound()

```
template<typename T1, typename T2>
smtrat::lra::Bound< T1, T2 >::~Bound ( )
```

15.47.5 Member Function Documentation

15.47.5.1 asConstraint()

```
template<typename T1, typename T2>
const FormulaT& smtrat::lra::Bound< T1, T2 >::asConstraint ( ) const [inline]
```

Returns

15.47.5.2 boundExists()

```
template<typename T1, typename T2>
void smtrat::lra::Bound< T1, T2 >::boundExists ( ) const [inline]
```

15.47.5.3 deduced()

```
template<typename T1, typename T2>
bool smtrat::lra::Bound< T1, T2 >::deduced ( ) const [inline]
```

Returns

15.47.5.4 exists()

```
template<typename T1, typename T2>
bool smtrat::lra::Bound< T1, T2 >::exists ( ) const [inline]
```

15.47.5.5 hasNeqRepresentation()

```
template<typename T1, typename T2>
bool smtrat::lra::Bound< T1, T2 >::hasNeqRepresentation ( ) const [inline]
```

Returns

15.47.5.6 isActive()

```
template<typename T1, typename T2>
bool smtrat::lra::Bound< T1, T2 >::isActive ( ) const [inline]
```

Returns

15.47.5.7 isComplementActive()

```
template<typename T1, typename T2>
bool smtrat::lra::Bound< T1, T2 >::isComplementActive ( ) const [inline]
```

Returns

15.47.5.8 isInfinite()

```
template<typename T1, typename T2>
bool smtrat::lra::Bound< T1, T2 >::isInfinite ( ) const  [inline]
```

Returns

15.47.5.9 isLowerBound()

```
template<typename T1, typename T2>
bool smtrat::lra::Bound< T1, T2 >::isLowerBound ( ) const  [inline]
```

Returns

15.47.5.10 isSatisfied()

```
template<typename T1, typename T2>
bool smtrat::lra::Bound< T1, T2 >::isSatisfied (
            const T1 & _delta ) const  [inline]
```

15.47.5.11 isUnassigned()

```
template<typename T1, typename T2>
bool smtrat::lra::Bound< T1, T2 >::isUnassigned ( ) const  [inline]
```

15.47.5.12 isUpperBound()

```
template<typename T1, typename T2>
bool smtrat::lra::Bound< T1, T2 >::isUpperBound ( ) const  [inline]
```

Returns

15.47.5.13 isWeak()

```
template<typename T1, typename T2>
bool smtrat::lra::Bound< T1, T2 >::isWeak ( ) const  [inline]
```

Returns

15.47.5.14 limit()

```
template<typename T1, typename T2>
const Value<T1>& smtrat::lra::Bound< T1, T2 >::limit ( ) const  [inline]
```

Returns

15.47.5.15 markAsDeleted()

```
template<typename T1, typename T2>
void smtrat::lra::Bound< T1, T2 >::markAsDeleted ( ) const  [inline]
```

Returns

15.47.5.16 markedAsDeleted()

```
template<typename T1, typename T2>
bool smtrat::lra::Bound< T1, T2 >::markedAsDeleted ( ) const [inline]
```

Returns

15.47.5.17 neqRepresentation()

```
template<typename T1, typename T2>
const FormulaT& smtrat::lra::Bound< T1, T2 >::neqRepresentation ( ) const [inline]
```

Returns

15.47.5.18 operator<() [1/3]

```
template<typename T1, typename T2>
bool smtrat::lra::Bound< T1, T2 >::operator< (
            const Bound< T1, T2 > & _bound ) const
```

Parameters

_bound	

Returns

15.47.5.19 operator<() [2/3]

```
template<typename T1, typename T2>
bool smtrat::lra::Bound< T1, T2 >::operator< (
            const T1 & _a ) const
```

Parameters

↩	
_↩	
a	

Returns

15.47.5.20 operator<() [3/3]

```
template<typename T1, typename T2>
bool smtrat::lra::Bound< T1, T2 >::operator< (
            const Value< T1 > & _value ) const
```

Parameters

_value	

Returns

15.47.5.21 operator<=() [1/2]

```
template<typename T1, typename T2>
bool smtrat::lra::Bound< T1, T2 >::operator<= (
            const T1 & _a ) const
```

Parameters

↩	
_↩	
a	

Returns

15.47.5.22 operator<=() [2/2]

```
template<typename T1, typename T2>
bool smtrat::lra::Bound< T1, T2 >::operator<= (
            const Value< T1 > & v ) const  [inline]
```

Returns

15.47.5.23 operator==() [1/2]

```
template<typename T1, typename T2>
bool smtrat::lra::Bound< T1, T2 >::operator== (
            const T1 & _a ) const
```

Parameters

↩ _a	

Returns

15.47.5.24 operator==() [2/2]

```
template<typename T1, typename T2>
bool smtrat::lra::Bound< T1, T2 >::operator== (
            const Value< T1 > & _value ) const
```

Parameters

_value	

Returns

15.47.5.25 operator>() [1/3]

```
template<typename T1, typename T2>
bool smtrat::lra::Bound< T1, T2 >::operator> (
            const Bound< T1, T2 > & _bound ) const
```

Parameters

_bound	

Returns

15.47.5.26 operator>() [2/3]

```
template<typename T1, typename T2>
bool smtrat::lra::Bound< T1, T2 >::operator> (
            const T1 & _a ) const
```

Parameters

↩ _↩ *a*	

Returns

15.47.5.27 operator>() [3/3]

```
template<typename T1, typename T2>
bool smtrat::lra::Bound< T1, T2 >::operator> (
            const Value< T1 > & _value ) const
```

Parameters

_value	

Returns

15.47.5.28 operator>=() [1/2]

```
template<typename T1, typename T2>
bool smtrat::lra::Bound< T1, T2 >::operator>= (
            const T1 & _a ) const
```

Parameters

↩ _↩ *a*	

Returns

15.47.5.29 operator>=() [2/2]

```
template<typename T1, typename T2>
bool smtrat::lra::Bound< T1, T2 >::operator>= (
            const Value< T1 > & v ) const  [inline]
```

Returns

15.47.5.30 origins()

```
template<typename T1, typename T2>
const std::vector<FormulaT>& smtrat::lra::Bound< T1, T2 >::origins ( ) const  [inline]
```

Returns

15.47.5.31 pInfo()

```
template<typename T1, typename T2>
Info* smtrat::lra::Bound< T1, T2 >::pInfo ( ) const [inline]
```

Returns

15.47.5.32 pLimit()

```
template<typename T1, typename T2>
const Value<T1>* smtrat::lra::Bound< T1, T2 >::pLimit ( ) const [inline]
```

Returns

15.47.5.33 pOrigins()

```
template<typename T1, typename T2>
const std::shared_ptr<std::vector<FormulaT> >& smtrat::lra::Bound< T1, T2 >::pOrigins ( )
const [inline]
```

Returns

15.47.5.34 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

_withOrigins	
_out	
_printTypebool	

15.47.5.35 pVariable()

```
template<typename T1, typename T2>
Variable<T1, T2>* smtrat::lra::Bound< T1, T2 >::pVariable ( ) const [inline]
```

Returns

15.47.5.36 resetNeqRepresentation()

```
template<typename T1, typename T2>
void smtrat::lra::Bound< T1, T2 >::resetNeqRepresentation ( ) const [inline]
```

15.47.5.37 setComplement()

```
template<typename T1, typename T2>
void smtrat::lra::Bound< T1, T2 >::setComplement (
            const Bound< T1, T2 > * _complement ) const  [inline]
```

15.47.5.38 setNeqRepresentation()

```
template<typename T1, typename T2>
void smtrat::lra::Bound< T1, T2 >::setNeqRepresentation (
            const FormulaT & _constraint ) const  [inline]
```

Parameters

_constraint	

15.47.5.39 toString()

```
template<typename T1, typename T2>
const std::string smtrat::lra::Bound< T1, T2 >::toString ( ) const
```

Returns

15.47.5.40 type()

```
template<typename T1, typename T2>
Type smtrat::lra::Bound< T1, T2 >::type ( ) const  [inline]
```

Returns

15.47.5.41 unmarkAsDeleted()

```
template<typename T1, typename T2>
void smtrat::lra::Bound< T1, T2 >::unmarkAsDeleted ( ) const  [inline]
```

Returns

15.47.5.42 variable()

```
template<typename T1, typename T2>
const Variable<T1, T2>& smtrat::lra::Bound< T1, T2 >::variable ( ) const  [inline]
```

Returns

15.47.6 Friends And Related Function Documentation

15.47.6.1 operator<<

```
template<typename T1, typename T2>
template<typename T3 , typename T4 >
std::ostream& operator<< (
            std::ostream & _ostream,
            const Bound< T3, T4 > & _bound )  [friend]
```

Parameters

_withOrigins	
_out	
_printTypebool	

15.48 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, carl::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.

15.48.1 Detailed Description

template<typename T>
class smtrat::vb::Bound< T >

Class for the bound of a variable.

15.48.2 Member Enumeration Documentation

15.48.2.1 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	

15.48.3 Constructor & Destructor Documentation

15.48.3.1 Bound() `template<typename T>`

```
smtrat::vb::Bound< T >::Bound (
            Rational *const _limit,
            Variable< T > *const _variable,
            Type _type )
```

Constructs this bound.

Parameters

_limit	A pointer to the limit (rational) of the bound. It is NULL, if the limit is not finite.
_variable	The variable to which this bound belongs.
_type	The type of the bound (either it is an equal bound or it is weak resp. strict and upper resp. lower)

15.48.3.2 ~Bound() `template<typename T>`

```
smtrat::vb::Bound< T >::~Bound ( )
```

Destructs this bound.

15.48.4 Member Function Documentation

15.48.4.1 activate() `template<typename T>`

```
bool smtrat::vb::Bound< T >::activate (
            const T & _origin ) const  [inline]
```

Adds an origin to this bound.

Parameters

_origin	The origin to add.

Returns true, if this has activated this bound; false, if the bound was already active before.

15.48.4.2 deactivate() `template<typename T>`

```
bool smtrat::vb::Bound< T >::deactivate (
            const T & _origin ) const  [inline]
```

Removes an origin from this bound.

Parameters

_origin	The origin to add.

Returns true, if this has deactivated this bound; false, if the bound was already inactive before.

15.48.4.3 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.

15.48.4.4 isInfinite()

```
template<typename T>
bool smtrat::vb::Bound< T >::isInfinite ( ) const [inline]
```

Returns true, if the bound infinite; false, otherwise.

15.48.4.5 isLowerBound()

```
template<typename T>
bool smtrat::vb::Bound< T >::isLowerBound ( ) const [inline]
```

Returns true, if the bound is a lower bound; false, otherwise.

15.48.4.6 isUpperBound()

```
template<typename T>
bool smtrat::vb::Bound< T >::isUpperBound ( ) const [inline]
```

Returns true, if the bound is an upper bound; false, otherwise.

15.48.4.7 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.

15.48.4.8 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

_bound	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.

15.48.4.9 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.

15.48.4.10 pLimit()

```
template<typename T>
const Rational* smtrat::vb::Bound< T >::pLimit ( ) const  [inline]
```

Returns A pointer to the limit of this bound.

15.48.4.11 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

_out	The stream to print on.
_withRelation	A flag indicating whether to print also a relation symbol in front of the bound value.

15.48.4.12 pVariable()

```
template<typename T>
Variable<T>* smtrat::vb::Bound< T >::pVariable ( ) const  [inline]
```

Returns A pointer to the variable wrapper considered by this bound.

15.48.4.13 type()

```
template<typename T>
Type smtrat::vb::Bound< T >::type ( ) const  [inline]
```

Returns The type of this bound.

15.48.4.14 variable()

```
template<typename T>
const Variable<T>& smtrat::vb::Bound< T >::variable ( ) const [inline]
```

Returns A constant reference to the variable wrapper considered by this bound.

15.48.5 Friends And Related Function Documentation

15.48.5.1 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

_out	The output stream to print on.
_bound	The bound to print.

Returns The output stream after printing the bound on it.

15.49 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)

15.49.1 Constructor & Destructor Documentation

15.49.1.1 Bound()

```
smtrat::mcsat::fm::Bound::Bound (
            ConstraintT constr,
            Poly p,
            Poly q,
            Rational r,
            bool neg ) [inline]
```

15.49.2 Friends And Related Function Documentation

15.49.2.1 operator<<

```
std::ostream& operator<< (
            std::ostream & os,
            const Bound & dt ) [friend]
```

15.49.3 Field Documentation

15.49.3.1 constr `ConstraintT smtrat::mcsat::fm::Bound::constr`

15.49.3.2 neg `bool smtrat::mcsat::fm::Bound::neg`

15.49.3.3 p `Poly smtrat::mcsat::fm::Bound::p`

15.49.3.4 q `Poly smtrat::mcsat::fm::Bound::q`

15.49.3.5 r `Rational smtrat::mcsat::fm::Bound::r`

15.50 smtrat::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).

15.50.1 Detailed Description Stores all necessary information of an branch, which can be used to detect probable infinite loop of branchings.

15.50.2 Constructor & Destructor Documentation

15.50.2.1 Branching()

```
smtrat::Branching::Branching (
    const typename Poly::PolyType & _polynomial,
    const Rational & _value ) [inline]
```

Constructor.

Parameters

_polynomial	The polynomial to branch at.
_value	The value to branch the polynomial at.

15.50.3 Field Documentation

15.50.3.1 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.

15.50.3.2 mPolynomial `Poly::PolyType smtrat::Branching::mPolynomial`

The polynomial to branch at.

15.50.3.3 mRepetitions `size_t smtrat::Branching::mRepetitions`

The number of repetitions of the branching.

15.50.3.4 mValue `Rational smtrat::Branching::mValue`

The value to branch the polynomial at.

15.51 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 isConstant () 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 & lowerBound () const
- const Integer & upperBound () 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)

15.51.1 Constructor & Destructor Documentation

15.51.1.1 BVAnnotation() [1/2] smtrat::BVAnnotation::BVAnnotation ( ) [inline]

15.51.1.2 BVAnnotation() [2/2] smtrat::BVAnnotation::BVAnnotation (
std::size_t *_width*,
bool *_signed*,
Integer *_offset* = *0*) [inline]

15.51.2 Member Function Documentation

15.51.2.1 bounds() const carl::Interval<Integer>& smtrat::BVAnnotation::bounds ( ) const [inline]

15.51.2.2 forProduct() static BVAnnotation smtrat::BVAnnotation::forProduct (
BVAnnotation *_factor1*,
BVAnnotation *_factor2*) [inline], [static]

15.51.2.3 forSum() static BVAnnotation smtrat::BVAnnotation::forSum (
BVAnnotation *_summand1*,
BVAnnotation *_summand2*) [inline], [static]

15.51.2.4 hasOffset() bool smtrat::BVAnnotation::hasOffset ( ) const [inline]

15.51.2.5 isConstant() bool smtrat::BVAnnotation::isConstant ( ) const [inline]

15.51.2.6 isSigned() bool smtrat::BVAnnotation::isSigned ( ) const [inline]

15.51.2.7 lowerBound() const Integer& smtrat::BVAnnotation::lowerBound ( ) const [inline]

15.51.2.8 offset() const Integer& smtrat::BVAnnotation::offset ( ) const [inline]

15.51.2.9 upperBound() const Integer& smtrat::BVAnnotation::upperBound ( ) const [inline]

15.51.2.10 width() std::size_t smtrat::BVAnnotation::width ( ) const [inline]

15.51.2.11 withOffset()

```
BVAnnotation smtrat::BVAnnotation::withOffset (
    Integer _newOffset ) const [inline]
```

15.51.2.12 withWidth()

```
BVAnnotation smtrat::BVAnnotation::withWidth (
    std::size_t _width ) const [inline]
```

15.51.3 Friends And Related Function Documentation

15.51.3.1 operator<<

```
std::ostream& operator<< (
    std::ostream & _out,
    const BVAnnotation & _type ) [friend]
```

15.52 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 ()

15.52.1 Constructor & Destructor Documentation

15.52.1.1 BVDirectEncoder()

```
smtrat::BVDirectEncoder::BVDirectEncoder ( ) [inline]
```

15.52.1.2 ~BVDirectEncoder()

```
smtrat::BVDirectEncoder::~BVDirectEncoder ( ) [inline]
```

15.52.2 Member Function Documentation

15.52.2.1 bitvectorBlastings()

```
const carl::FastMap<BitVec, Variables> smtrat::BVDirectEncoder↩
::bitvectorBlastings ( ) const [inline]
```

15.52.2.2 encode()

```
const FormulaSetT& smtrat::BVDirectEncoder::encode (
    const FormulaT & _inputFormula ) [inline]
```

15.52.2.3 introducedVariables()

```
const std::set<carl::Variable>& smtrat::BVDirectEncoder::introduced↩
Variables ( ) const [inline]
```

15.53 smtrat::BVModule< Settings > Class Template Reference

```
#include <BVModule.h>
```

Public Types

- typedef Settings SettingsType
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- BVModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=NULL)

  Constructors.
- ~BVModule ()

  Destructor.
- 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 hasValidInfeasibleSubset () 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

- 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.

- 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 &_branching↩ Value) 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 _preferLeft↩ Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩ ::vector< FormulaT >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩ AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩ CaseWeak=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 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.

15.53.1 Member Typedef Documentation

15.53.1.1 SettingsType

```
template<typename Settings >
typedef Settings smtrat::BVModule< Settings >::SettingsType
```

15.53.2 Member Enumeration Documentation

15.53.2.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

15.53.3 Constructor & Destructor Documentation

15.53.3.1 BVModule()

```
template<class Settings >
smtrat::BVModule< Settings >::BVModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = NULL )
```

Constructors.

15.53.3.2 ~BVModule()

```
template<class Settings >
smtrat::BVModule< Settings >::~BVModule ( )
```

Destructor.

15.53.4 Member Function Documentation

15.53.4.1 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.

15.53.4.2 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 >.

15.53.4.3 addCore()

```
template<class 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

_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.

15.53.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
            const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.53.4.5 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.

15.53.4.6 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.

15.53.4.7 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.

15.53.4.8 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.

15.53.4.9 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.

15.53.4.10 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

_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.

15.53.4.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns All satisfying assignments, if existent.

15.53.4.12 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.

15.53.4.13 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.

15.53.4.14 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.

15.53.4.15 branchAt() [1/6]

```
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]
```

15.53.4.16 branchAt() [2/6]

```
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]
```

15.53.4.17 branchAt() [3/6]

```
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]
```

15.53.4.18 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.53.4.19 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            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]
```

15.53.4.20 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

15.53.4.21 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in smtrat::PModule.

15.53.4.22 checkCore()

```
template<class 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.

15.53.4.23 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
    std::vector< FormulaSetT >::const_iterator _infsubset,
    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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.53.4.24 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.

15.53.4.25 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.53.4.26 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.53.4.27 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.53.4.28 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.53.4.29 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.53.4.30 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

15.53.4.31 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

_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.

15.53.4.32 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.53.4.33 constraintsToInform()

```
const carl::FastSet<FormulaT>& smtrat::Module::constraintsTo↩
Inform ( ) const [inline], [inherited]
```

Returns The constraints which the backends must be informed about.

15.53.4.34 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

15.53.4.35 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)

15.53.4.36 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

_constraint	The constraint to remove from internal data structures.

15.53.4.37 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.53.4.38 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.53.4.39 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.53.4.40 evaluateBVFormula()

```
template<class Settings >
size_t smtrat::BVModule< Settings >::evaluateBVFormula (
    const FormulaT & formula ) [protected]
```

15.53.4.41 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceived↩
VariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.53.4.42 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBest↩
Origin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.53.4.43 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.

15.53.4.44 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.

15.53.4.45 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

15.53.4.46 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

15.53.4.47 getBackendsAllModels()

`void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

15.53.4.48 getBackendsModel()

`void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

15.53.4.49 getInfeasibleSubsets() [1/2]

`void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

15.53.4.50 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.53.4.51 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.53.4.52 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.

15.53.4.53 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.53.4.54 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.53.4.55 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.53.4.56 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 reveived formula, if the Boolean is true.

Reimplemented in smtrat::PModule.

15.53.4.57 hasLemmas()

```
bool smtrat::Module::hasLemmas ( ) [inline], [inherited]
```

Checks whether this module or any of its backends provides any lemmas.

15.53.4.58 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).

15.53.4.59 id()

```
std::size_t smtrat::Module::id ( ) const [inline], [inherited]
```

Returns A unique ID to identify this module instance.

15.53.4.60 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.

15.53.4.61 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.53.4.62 informBackends()

```
void smtrat::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.

15.53.4.63 informCore()

```
template<class Settings >
bool smtrat::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

_constraint	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.

15.53.4.64 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.53.4.65 init()

```
template<class Settings >
void smtrat::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 smtrat::Module.

15.53.4.66 is_minimizing()

```
bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]
```

15.53.4.67 isLemmaLevel()

```
bool smtrat::Module::isLemmaLevel (
    LemmaLevel level ) [protected], [inherited]
```

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.53.4.68 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().

15.53.4.69 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.

15.53.4.70 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.53.4.71 model()

```
const Model& smtrat::Module::model ( ) const [inline], [inherited]
```

Returns The assignment of the current satisfiable result, if existent.

15.53.4.72 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.

15.53.4.73 moduleName()

```
template<typename Settings >
std::string smtrat::BVModule< Settings >::moduleName ( ) const [inline], [virtual]
```

Returns The name of the given module type as name.

Reimplemented from smtrat::Module.

15.53.4.74 objective()

```
carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]
```

15.53.4.75 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
    const FormulaT & _origin ) const [protected], [inherited]
```

15.53.4.76 passedFormulaBegin()

```
ModuleInput::iterator smtrat::Module::passedFormulaBegin ( )
[inline], [protected], [inherited]
```

Returns An iterator to the end of the passed formula. TODO: disable this method

15.53.4.77 passedFormulaEnd()

```
ModuleInput::iterator smtrat::Module::passedFormulaEnd ( )
[inline], [protected], [inherited]
```

Returns An iterator to the end of the passed formula.

15.53.4.78 pPassedFormula()

```
const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline],
[inherited]
```

Returns A pointer to the formula passed to the backends of this module.

15.53.4.79 pReceivedFormula()

```
const ModuleInput* smtrat::Module::pReceivedFormula ( ) const
[inline], [inherited]
```

Returns A pointer to the formula passed to this module.

15.53.4.80 print()

```
void smtrat::Module::print (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.53.4.81 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

_out	The stream to print the assignment on.

15.53.4.82 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.53.4.83 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

_out	The stream to print the assignment on.

15.53.4.84 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.53.4.85 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.53.4.86 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.

15.53.4.87 receivedFormulaChecked()

```
void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]
```

Notifies that the received formulas has been checked.

15.53.4.88 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAsInfeasible↩
Subset (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

15.53.4.89 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const [inline], [inherited]
```

15.53.4.90 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.

15.53.4.91 removeCore()

```
template<class Settings >
void smtrat::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 smtrat::Module.

15.53.4.92 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin (
            ModuleInput::iterator _formula,
            const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.53.4.93 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins
(
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

15.53.4.94 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline],
[inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.53.4.95 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.53.4.96 runBackends() [1/2]

`virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]`

Reimplemented in smtrat::PModule.

15.53.4.97 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

_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 in smtrat::PModule.

15.53.4.98 setId()

```
void smtrat::Module::setId (
    std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.53.4.99 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.

15.53.4.100 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.

15.53.4.101 splitUnequalConstraint()

`void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.53.4.102 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.53.4.103 transferBackendModel()

`template<class Settings > void smtrat::BVModule< Settings >::transferBackendModel ( ) const [protected]`

15.53.4.104 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 >.

15.53.4.105 updateLemmas()

`void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.53.4.106 updateModel()

`template<class Settings > void smtrat::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 smtrat::Module.

15.53.4.107 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).

15.53.5 Field Documentation

15.53.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.53.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.53.5.3 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.

15.53.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.53.5.5 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

15.53.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.53.5.7 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.

15.53.5.8 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.

15.53.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.53.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.53.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.53.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.53.5.13 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.

15.53.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.53.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.53.5.16 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.53.5.17 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.

15.53.5.18 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]`

Reusable splitting variables.

15.53.5.19 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.53.5.20 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.53.5.21 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.

15.53.5.22 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

15.53.5.23 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).

15.53.5.24 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.54 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

15.54.1 Detailed Description

template<typename IterType>
struct smtrat::by_address_hasher< IterType >

Hash an iterator by the pointed-to address.

15.54.2 Member Typedef Documentation

15.54.2.1 pointer_type `template<typename IterType >`

```
typedef std::iterator_traits<IterType>::pointer smtrat::by_address_hasher< IterType >::pointer_type
```

15.54.3 Member Function Documentation

15.54.3.1 operator()() `template<typename IterType >`

```
std::size_t smtrat::by_address_hasher< IterType >::operator() (
            const IterType & iter ) const [inline], [noexcept]
```

15.55 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 ConstraintIt >
  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

15.55.1 Member Typedef Documentation

15.55.1.1 SettingsT

```
template<typename Settings>
using smtrat::cad::CAD< Settings >::SettingsT = Settings
```

15.55.2 Constructor & Destructor Documentation

15.55.2.1 CAD()

```
template<typename Settings>
smtrat::cad::CAD< Settings >::CAD ( ) [inline]
```

15.55.2.2 ~CAD()

```
template<typename Settings>
smtrat::cad::CAD< Settings >::~CAD ( ) [inline]
```

15.55.3 Member Function Documentation

15.55.3.1 addConstraint()

```
template<typename Settings>
void smtrat::cad::CAD< Settings >::addConstraint (
            const ConstraintT & c ) [inline]
```

15.55.3.2 check()

```
template<typename Settings>
Answer smtrat::cad::CAD< Settings >::check (
            Assignment & assignment,
            std::vector< FormulaSetT > & mis ) [inline]
```

15.55.3.3 checkFullSamples()

```
template<typename Settings>
Answer smtrat::cad::CAD< Settings >::checkFullSamples (
            Assignment & assignment ) [inline]
```

15.55.3.4 checkPartialSample()

```
template<typename Settings>
template<typename Iterator >
Answer smtrat::cad::CAD< Settings >::checkPartialSample (
            Iterator & it,
            std::size_t level ) [inline]
```

15.55.3.5 dim()

```
template<typename Settings>
std::size_t smtrat::cad::CAD< Settings >::dim ( ) const [inline]
```

15.55.3.6 enumerateSolutions()

```
template<typename Settings>
std::vector<Assignment> smtrat::cad::CAD< Settings >::enumerateSolutions ( ) [inline]
```

15.55.3.7 evaluateSample()

```
template<typename Settings>
template<typename ConstraintIt >
bool smtrat::cad::CAD< Settings >::evaluateSample (
            Sample & sample,
            const ConstraintIt & constraint,
            Assignment & assignment ) const [inline]
```

15.55.3.8 generateConflictGraph()

```
template<typename Settings>
ConflictGraph smtrat::cad::CAD< Settings >::generateConflictGraph ( ) const [inline]
```

15.55.3.9 generateCovering()

```
template<typename Settings>
auto smtrat::cad::CAD< Settings >::generateCovering ( ) const [inline]
```

15.55.3.10 getBounds()

```
template<typename Settings>
const auto& smtrat::cad::CAD< Settings >::getBounds ( ) const [inline]
```

15.55.3.11 getConstraintMap()

```
template<typename Settings>
const auto& smtrat::cad::CAD< Settings >::getConstraintMap ( ) const [inline]
```

15.55.3.12 getConstraints()

```
template<typename Settings>
const auto& smtrat::cad::CAD< Settings >::getConstraints ( ) const [inline]
```

15.55.3.13 getLifting()

```
template<typename Settings>
const auto& smtrat::cad::CAD< Settings >::getLifting ( ) const [inline]
```

15.55.3.14 getProjection()

```
template<typename Settings>
const auto& smtrat::cad::CAD< Settings >::getProjection ( ) const [inline]
```

15.55.3.15 getVariables()

```
template<typename Settings>
const auto& smtrat::cad::CAD< Settings >::getVariables ( ) const [inline]
```

15.55.3.16 isIdValid()

```
template<typename Settings>
bool smtrat::cad::CAD< Settings >::isIdValid (
            std::size_t id ) const [inline]
```

15.55.3.17 removeConstraint()

```
template<typename Settings>
void smtrat::cad::CAD< Settings >::removeConstraint (
            const ConstraintT & c ) [inline]
```

15.55.3.18 reset()

```
template<typename Settings>
void smtrat::cad::CAD< Settings >::reset (
            const std::vector< carl::Variable > & vars ) [inline]
```

15.55.4 Friends And Related Function Documentation

15.55.4.1 CADCore `template<typename Settings>`

```
template<CoreHeuristic CH>
friend struct CADCore [friend]
```

15.55.5 Field Documentation

15.55.5.1 thp `template<typename Settings>`

```
debug::TikzHistoryPrinter smtrat::cad::CAD< Settings >::thp
```

15.56 smtrat::qe::cad::CAD< Settings > Class Template Reference

```
#include <CAD.h>
```

Public Types

- using TreeIterator = 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 ()

15.56.1 Member Typedef Documentation

15.56.1.1 TreeIterator `template<typename Settings>`

```
using smtrat::qe::cad::CAD< Settings >::TreeIterator = typename smtrat::cad::LiftingTree<Settings>↩
::Iterator
```

15.56.2 Constructor & Destructor Documentation

15.56.2.1 CAD() `template<typename Settings>`

```
smtrat::qe::cad::CAD< Settings >::CAD ( ) [inline]
```

15.56.3 Member Function Documentation

15.56.3.1 addConstraint() `template<typename Settings>`

```
void smtrat::qe::cad::CAD< Settings >::addConstraint (
            const ConstraintT & c ) [inline]
```

15.56.3.2 addPolynomial() `template<typename Settings>`

```
void smtrat::qe::cad::CAD< Settings >::addPolynomial (
            const Poly & p ) [inline]
```

15.56.3.3 dim() `template<typename Settings>`

```
std::size_t smtrat::qe::cad::CAD< Settings >::dim ( ) const [inline]
```

15.56.3.4 getLifting()

```
template<typename Settings>
const auto& smtrat::qe::cad::CAD< Settings >::getLifting ( ) const [inline]
```

15.56.3.5 getProjection()

```
template<typename Settings>
const auto& smtrat::qe::cad::CAD< Settings >::getProjection ( ) const [inline]
```

15.56.3.6 lift()

```
template<typename Settings>
void smtrat::qe::cad::CAD< Settings >::lift ( ) [inline]
```

15.56.3.7 project()

```
template<typename Settings>
void smtrat::qe::cad::CAD< Settings >::project ( ) [inline]
```

15.56.3.8 removeConstraint()

```
template<typename Settings>
void smtrat::qe::cad::CAD< Settings >::removeConstraint (
            const ConstraintT & c ) [inline]
```

15.56.3.9 removePolynomial()

```
template<typename Settings>
void smtrat::qe::cad::CAD< Settings >::removePolynomial (
            std::size_t level,
            std::size_t id ) [inline]
```

15.56.3.10 reset()

```
template<typename Settings>
void smtrat::qe::cad::CAD< Settings >::reset (
            const std::vector< carl::Variable > & vars ) [inline]
```

15.57 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 ConstraintIts = 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 > mConstraintIts
- 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)

15.57.1 Member Typedef Documentation

15.57.1.1 Callback `template<Backtracking BT>`

```
using smtrat::cad::CADConstraints< BT >::Callback = std::function<void(const UPoly&, std↩
::size_t, bool)>
```

15.57.1.2 ConstraintIts `template<Backtracking BT>`

```
using smtrat::cad::CADConstraints< BT >::ConstraintIts = std::vector<typename ConstraintMap↩
::iterator>  [protected]
```

15.57.1.3 ConstraintMap `template<Backtracking BT>`

```
using smtrat::cad::CADConstraints< BT >::ConstraintMap = std::map<ConstraintT, std::size_t,
ConstraintComparator>  [protected]
```

15.57.1.4 VariableBounds `template<Backtracking BT>`

```
using smtrat::cad::CADConstraints< BT >::VariableBounds = vb::VariableBounds<ConstraintT>
```

15.57.2 Constructor & Destructor Documentation

15.57.2.1 CADConstraints() [1/2] `template<Backtracking BT>`

```
smtrat::cad::CADConstraints< BT >::CADConstraints (
            const Callback & onAdd,
            const Callback & onAddEq,
            const Callback & onRemove )  [inline]
```

15.57.2.2 CADConstraints() [2/2]

```
template<Backtracking BT>
smtrat::cad::CADConstraints< BT >::CADConstraints (
            const CADConstraints< BT > & ) [delete]
```

15.57.3 Member Function Documentation

15.57.3.1 add()

```
template<Backtracking BT>
std::size_t smtrat::cad::CADConstraints< BT >::add (
            const ConstraintT & c ) [inline]
```

15.57.3.2 bounds()

```
template<Backtracking BT>
const auto& smtrat::cad::CADConstraints< BT >::bounds ( ) const [inline]
```

15.57.3.3 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]
```

15.57.3.4 checkForTrivialConflict()

```
template<Backtracking BT>
bool smtrat::cad::CADConstraints< BT >::checkForTrivialConflict (
            std::vector< FormulaSetT > & mis ) const [inline]
```

15.57.3.5 exportAsDot()

```
template<Backtracking BT>
void smtrat::cad::CADConstraints< BT >::exportAsDot (
            std::ostream & out ) const [inline]
```

15.57.3.6 indexed()

```
template<Backtracking BT>
const auto& smtrat::cad::CADConstraints< BT >::indexed ( ) const [inline]
```

15.57.3.7 level()

```
template<Backtracking BT>
std::size_t smtrat::cad::CADConstraints< BT >::level (
            std::size_t id ) const [inline]
```

15.57.3.8 operator[]()

```
template<Backtracking BT>
const ConstraintT& smtrat::cad::CADConstraints< BT >::operator[] (
            std::size_t id ) const [inline]
```

15.57.3.9 ordered()

```
template<Backtracking BT>
const auto& smtrat::cad::CADConstraints< BT >::ordered ( ) const [inline]
```

15.57.3.10 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.

15.57.3.11 reset() template<Backtracking BT>

```
void smtrat::cad::CADConstraints< BT >::reset (
            const std::vector< carl::Variable > & vars )  [inline]
```

15.57.3.12 size() template<Backtracking BT>

```
std::size_t smtrat::cad::CADConstraints< BT >::size ( ) const  [inline]
```

15.57.3.13 unsatByBounds() template<Backtracking BT>

```
const auto& smtrat::cad::CADConstraints< BT >::unsatByBounds ( ) const  [inline]
```

15.57.3.14 valid() template<Backtracking BT>

```
bool smtrat::cad::CADConstraints< BT >::valid (
            std::size_t id ) const  [inline]
```

15.57.3.15 vars() template<Backtracking BT>

```
const std::vector<carl::Variable>& smtrat::cad::CADConstraints< BT >::vars ( ) const  [inline]
```

15.57.4 Friends And Related Function Documentation

15.57.4.1 operator<< template<Backtracking BT>

```
template<Backtracking B>
std::ostream& operator<< (
            std::ostream & os,
            const CADConstraints< B > & cc )  [friend]
```

15.57.5 Field Documentation

15.57.5.1 mActiveConstraintMap template<Backtracking BT>

```
ConstraintMap smtrat::cad::CADConstraints< BT >::mActiveConstraintMap  [protected]
```

15.57.5.2 mAddCallback template<Backtracking BT>

```
Callback smtrat::cad::CADConstraints< BT >::mAddCallback  [protected]
```

15.57.5.3 mAddEqCallback template<Backtracking BT>

```
Callback smtrat::cad::CADConstraints< BT >::mAddEqCallback  [protected]
```

15.57.5.4 mBounds template<Backtracking BT>

VariableBounds smtrat::cad::CADConstraints< BT >::mBounds [protected]

15.57.5.5 mConstraintIts template<Backtracking BT>

std::vector<typename ConstraintMap::iterator> smtrat::cad::CADConstraints< BT >::mConstraint↩
Its [protected]

15.57.5.6 mConstraintLevels template<Backtracking BT>

std::vector<std::size_t> smtrat::cad::CADConstraints< BT >::mConstraintLevels [protected]

15.57.5.7 mConstraintMap template<Backtracking BT>

ConstraintMap smtrat::cad::CADConstraints< BT >::mConstraintMap [protected]

15.57.5.8 mIDPool template<Backtracking BT>

carl::IDPool smtrat::cad::CADConstraints< BT >::mIDPool [protected]

15.57.5.9 mRemoveCallback template<Backtracking BT>

Callback smtrat::cad::CADConstraints< BT >::mRemoveCallback [protected]

15.57.5.10 mSatByBounds template<Backtracking BT>

carl::Bitset smtrat::cad::CADConstraints< BT >::mSatByBounds [protected]

List of constraints that are satisfied by bounds.

15.57.5.11 mUnsatByBounds template<Backtracking BT>

carl::Bitset smtrat::cad::CADConstraints< BT >::mUnsatByBounds [protected]

List of constraints that are infeasible due to bounds.

15.57.5.12 mVariables template<Backtracking BT>

std::vector<carl::Variable> smtrat::cad::CADConstraints< BT >::mVariables [protected]

15.58 smtrat::cad::CADCore< CH > Struct Template Reference

#include <CADCore.h>

15.59 smtrat::cad::CADCore< CoreHeuristic::BySample > Struct Template Reference

#include <CADCore.h>

Public Member Functions

- template<typename CAD >
  Answer operator() (Assignment &assignment, CAD &cad)

15.59.1 Member Function Documentation

15.59.1.1 operator()() template<typename CAD >

```
Answer smtrat::cad::CADCore< CoreHeuristic::BySample >::operator() (
            Assignment & assignment,
            CAD & cad )  [inline]
```

15.60 smtrat::cad::CADCore< CoreHeuristic::EnumerateAll > Struct Template Reference

```
#include <CADCore.h>
```

Public Member Functions

- template<typename CAD >
  Answer operator() (Assignment &assignment, CAD &cad)

15.60.1 Member Function Documentation

15.60.1.1 operator()() template<typename CAD >

```
Answer smtrat::cad::CADCore< CoreHeuristic::EnumerateAll >::operator() (
            Assignment & assignment,
            CAD & cad )  [inline]
```

15.61 smtrat::cad::CADCore< CoreHeuristic::Interleave > Struct Template 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)

15.61.1 Member Function Documentation

15.61.1.1 doLifting() template<typename CAD >

```
bool smtrat::cad::CADCore< CoreHeuristic::Interleave >::doLifting (
            CAD & cad )  [inline]
```

15.61.1.2 doProjection() template<typename CAD >

```
bool smtrat::cad::CADCore< CoreHeuristic::Interleave >::doProjection (
            CAD & cad )  [inline]
```

15.61.1.3 operator()() template<typename CAD >

```
Answer smtrat::cad::CADCore< CoreHeuristic::Interleave >::operator() (
            Assignment & assignment,
            CAD & cad )  [inline]
```

15.61.1.4 preferLifting()

```
template<typename It >
bool smtrat::cad::CADCore< CoreHeuristic::Interleave >::preferLifting (
            const It & it ) [inline]
```

15.62 smtrat::cad::CADCore< CoreHeuristic::PreferProjection > Struct Template Reference

```
#include <CADCore.h>
```

Public Member Functions

- template<typename CAD >
  Answer operator() (Assignment &assignment, CAD &cad)

15.62.1 Member Function Documentation

15.62.1.1 operator()()

```
template<typename CAD >
Answer smtrat::cad::CADCore< CoreHeuristic::PreferProjection >::operator() (
            Assignment & assignment,
            CAD & cad ) [inline]
```

15.63 smtrat::cad::CADCore< CoreHeuristic::PreferSampling > Struct Template Reference

```
#include <CADCore.h>
```

Public Member Functions

- template<typename CAD >
  Answer operator() (Assignment &assignment, CAD &cad)

15.63.1 Member Function Documentation

15.63.1.1 operator()()

```
template<typename CAD >
Answer smtrat::cad::CADCore< CoreHeuristic::PreferSampling >::operator() (
            Assignment & assignment,
            CAD & cad ) [inline]
```

15.64 smtrat::qe::cad::CADElimination Class Reference

```
#include <CADElimination.h>
```

Public Member Functions

- CADElimination (const FormulaT &quantifierFreePart, const QEQuery &quantifiers)
- FormulaT eliminateQuantifiers ()

15.64.1 Constructor & Destructor Documentation

15.64.1.1 CADElimination()

```
smtrat::qe::cad::CADElimination::CADElimination (
            const FormulaT & quantifierFreePart,
            const QEQuery & quantifiers )
```

15.64.2 Member Function Documentation

15.64.2.1 eliminateQuantifiers()

```
FormulaT smtrat::qe::cad::CADElimination::eliminateQuantifiers (
)
```

15.65 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)

15.65.1 Constructor & Destructor Documentation

15.65.1.1 CADPreprocessor()

```
smtrat::cad::CADPreprocessor::CADPreprocessor (
    const std::vector< carl::Variable > & vars ) [inline]
```

15.65.2 Member Function Documentation

15.65.2.1 addConstraint()

```
void smtrat::cad::CADPreprocessor::addConstraint (
    const ConstraintT & c )
```

15.65.2.2 equalities()

```
const auto& smtrat::cad::CADPreprocessor::equalities ( ) const [inline]
```

15.65.2.3 getConflict()

```
std::set< FormulaT > smtrat::cad::CADPreprocessor::getConflict ( )
const
```

15.65.2.4 inequalities()

```
const auto& smtrat::cad::CADPreprocessor::inequalities ( ) const [inline]
```

15.65.2.5 model()

```
const Model& smtrat::cad::CADPreprocessor::model ( ) const [inline]
```

15.65.2.6 postprocessConflict()

```
void smtrat::cad::CADPreprocessor::postprocessConflict (
    std::set< FormulaT > & mis ) const
```

15.65.2.7 preprocess() `bool smtrat::cad::CADPreprocessor::preprocess ( )`

Performs the preprocessing.

Return false if a direct conflict was found, true otherwise.

15.65.2.8 removeConstraint() `void smtrat::cad::CADPreprocessor::removeConstraint ( const ConstraintT & c )`

15.65.2.9 result() `template<typename Map > preprocessor::ConstraintUpdate smtrat::cad::CADPreprocessor::result ( const Map & oldC ) const [inline]`

15.65.2.10 simplify() `FormulaT smtrat::cad::CADPreprocessor::simplify ( const FormulaT & f ) const [inline]`

15.65.3 Friends And Related Function Documentation

15.65.3.1 operator<< `std::ostream& operator<< ( std::ostream & os, const CADPreprocessor & cadpp ) [friend]`

15.66 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()

15.66.1 Member Function Documentation

15.66.1.1 register_hook() `static bool smtrat::cad::CADPreprocessorSettings::register_hook ( ) [inline], [static]`

15.66.1.2 register_settings() `static void smtrat::cad::CADPreprocessorSettings::register_settings ( SettingsParser & parser ) [inline], [static]`

15.66.2 Field Documentation

15.66.2.1 disable_resultants `bool smtrat::cad::CADPreprocessorSettings::disable_resultants = false`

15.66.2.2 disable_variable_elimination

`bool smtrat::cad::CADPreprocessorSettings::disable_variable↩ _elimination = false`

15.66.2.3 dummy

`const bool smtrat::cad::CADPreprocessorSettings::dummy = CADPreprocessorSettings::register_h [static]`

15.67 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

15.67.1 Field Documentation

15.67.1.1 backtracking

`constexpr smtrat::cad::Backtracking smtrat::qe::cad::CADSettings↩ ::backtracking = smtrat::cad::Backtracking::UNORDERED [static], [constexpr]`

15.67.1.2 coreHeuristic

`constexpr CoreHeuristic smtrat::cad::BaseSettings::coreHeuristic = cad::CoreHeuristic::PreferProjection [static], [constexpr], [inherited]`

15.67.1.3 fullSampleComparator

`constexpr FullSampleCompareStrategy smtrat::cad::BaseSettings↩ ::fullSampleComparator = cad::FullSampleCompareStrategy::Default [static], [constexpr], [inherited]`

15.67.1.4 incrementality

`constexpr smtrat::cad::Incrementality smtrat::qe::cad::CADSettings↩ ::incrementality = smtrat::cad::Incrementality::NONE [static], [constexpr]`

15.67.1.5 misHeuristic

`constexpr MISHeuristic smtrat::cad::BaseSettings::misHeuristic = cad::MISHeuristic::GRE [static], [constexpr], [inherited]`

15.67.1.6 projectionComparator

`constexpr ProjectionCompareStrategy smtrat::cad::BaseSettings↩ ::projectionComparator = cad::ProjectionCompareStrategy::Default [static], [constexpr], [inherited]`

15.67.1.7 projectionOperator

`constexpr smtrat::cad::ProjectionType smtrat::qe::cad::CADSettings↩ ::projectionOperator = smtrat::cad::ProjectionType::Brown [static], [constexpr]`

15.67.1.8 sampleComparator

`constexpr SampleCompareStrategy smtrat::cad::BaseSettings::sampleComparator = cad::SampleCompareStrategy::Default [static], [constexpr], [inherited]`

15.67.1.9 simplifyProjectionByBounds

`constexpr bool smtrat::cad::BaseSettings::simplifyProjectionByBounds = true [static], [constexpr], [inherited]`

15.67.1.10 trivialSampleRadius

`constexpr std::size_t smtrat::cad::BaseSettings::trivialSampleRadius = 1 [static], [constexpr], [inherited]`

15.68 smtrat::CardinalityEncoder Class Reference

`#include <CardinalityEncoder.h>`

Public Member Functions

- CardinalityEncoder ()
- Rational encodingSize (const ConstraintT &constraint)
- bool canEncode (const ConstraintT &constraint)
- std::string name ()
- boost::optional< FormulaT > encode (const ConstraintT &constraint)

  Encodes an arbitrary constraint.

Data Fields

- std::size_t problem_size

Protected Member Functions

- boost::optional< FormulaT > doEncode (const ConstraintT &constraint)
- FormulaT generateVarChain (const std::set< carl::Variable > &vars, carl::FormulaType type)

15.68.1 Constructor & Destructor Documentation

15.68.1.1 CardinalityEncoder()

`smtrat::CardinalityEncoder::CardinalityEncoder ( ) [inline]`

15.68.2 Member Function Documentation

15.68.2.1 canEncode()

`bool smtrat::CardinalityEncoder::canEncode ( const ConstraintT & constraint ) [virtual]`

Implements smtrat::PseudoBoolEncoder.

15.68.2.2 doEncode()

`boost::optional< FormulaT > smtrat::CardinalityEncoder::doEncode ( const ConstraintT & constraint ) [protected], [virtual]`

Implements smtrat::PseudoBoolEncoder.

15.68.2.3 encode()

`boost::optional< FormulaT > smtrat::PseudoBoolEncoder::encode ( const ConstraintT & constraint ) [inherited]`

Encodes an arbitrary constraint.

Returns encoded formula

15.68.2.4 encodingSize()

```
Rational smtrat::CardinalityEncoder::encodingSize (
    const ConstraintT & constraint ) [virtual]
```

Reimplemented from smtrat::PseudoBoolEncoder.

15.68.2.5 generateVarChain()

```
FormulaT smtrat::PseudoBoolEncoder::generateVarChain (
    const std::set< carl::Variable > & vars,
    carl::FormulaType type ) [protected], [inherited]
```

15.68.2.6 name()

```
std::string smtrat::CardinalityEncoder::name ( ) [inline], [virtual]
```

Reimplemented from smtrat::PseudoBoolEncoder.

15.68.3 Field Documentation

15.68.3.1 problem_size

```
std::size_t smtrat::PseudoBoolEncoder::problem_size [inherited]
```

15.69 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)

15.69.1 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.

15.69.2 Member Function Documentation

15.69.2.1 compute()

```
template<CellHeuristic H>
template<typename T >
static std::optional<datastructures::CellRepresentation<T> > smtrat::cadcells::representation::cell<
H >::compute (
    datastructures::SampledDerivationRef< T > & der ) [static]
```

15.70 smtrat::cadcells::representation::cell< CellHeuristic::BIGGEST_CELL > Struct Template Reference

```
#include <heuristics_cell.h>
```

Static Public Member Functions

- template<typename T >
  static std::optional< datastructures::CellRepresentation< T > > compute (datastructures::SampledDerivationRef< T > &der)

15.70.1 Member Function Documentation

15.70.1.1 compute()

```
template<typename T >
static std::optional<datastructures::CellRepresentation<T> > smtrat::cadcells::representation::cell<
CellHeuristic::BIGGEST_CELL >::compute (
            datastructures::SampledDerivationRef< T > & der ) [inline], [static]
```

15.71 smtrat::cadcells::representation::cell< CellHeuristic::CHAIN_EQ > Struct Template Reference

```
#include <heuristics_cell.h>
```

Static Public Member Functions

- template<typename T >
  static std::optional< datastructures::CellRepresentation< T > > compute (datastructures::SampledDerivationRef< T > &der)

15.71.1 Member Function Documentation

15.71.1.1 compute()

```
template<typename T >
static std::optional<datastructures::CellRepresentation<T> > smtrat::cadcells::representation::cell<
CellHeuristic::CHAIN_EQ >::compute (
            datastructures::SampledDerivationRef< T > & der ) [inline], [static]
```

15.72 smtrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS > Struct Template Reference

```
#include <heuristics_cell.h>
```

Static Public Member Functions

- template<typename T >
  static std::optional< datastructures::CellRepresentation< T > > compute (datastructures::SampledDerivationRef< T > &der)

15.72.1 Member Function Documentation

15.72.1.1 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]
```

15.73 smtrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS_EQ > Struct Template Reference

```
#include <heuristics_cell.h>
```

Static Public Member Functions

- template<typename T >
  static std::optional< datastructures::CellRepresentation< T > > compute (datastructures::SampledDerivationRef< T > &der)

15.73.1 Member Function Documentation

15.73.1.1 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]
```

15.74 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

15.74.1 Member Function Documentation

15.74.1.1 hash_on_level()

```
std::size_t smtrat::cadcells::operators::properties::cell_connected↩
::hash_on_level ( ) const [inline]
```

15.74.1.2 level()

```
std::size_t smtrat::cadcells::operators::properties::cell_connected::level ( )
const [inline]
```

15.74.2 Field Documentation

15.74.2.1 is_flag

```
constexpr bool smtrat::cadcells::operators::properties::cell_connected::is_↩
flag = true [static], [constexpr]
```

15.74.2.2 lvl

```
std::size_t smtrat::cadcells::operators::properties::cell_connected::lvl
```

15.75 smtrat::cadcells::datastructures::CellDescription Class Reference

Holds the description of a cell.

```
#include <roots.h>
```

Public Member Functions

- CellDescription (IndexedRoot bound)
- CellDescription (IndexedRoot lower, IndexedRoot upper)
- CellDescription (Bound, IndexedRoot upper)
- CellDescription (IndexedRoot lower, Bound)
- CellDescription (Bound, Bound)
- CellDescription ()
- bool is_sector () const
- bool is_section () const
- const IndexedRoot & section_defining () const

  In case of a section, the defining indexed root is returned.

- const auto & lower () const

  Returns the lower bound as IndexedRoot if finite or std::nullopt if the lower bound is -oo.

- const auto & upper () const

  Returns the upper bound as IndexedRoot if finite or std::nullopt if the upper bound is oo.

- std::optional< IndexedRoot > lower_defining () const
- std::optional< IndexedRoot > upper_defining () const
- boost::container::flat_set< PolyRef > polys () const

15.75.1 Detailed Description

Holds the description of a cell.

A cell is either a section [xi,xi] or a section (-00,xi), (xi, xi'), (xi, oo) where xi,xi' are indexed roots.

15.75.2 Constructor & Destructor Documentation

15.75.2.1 CellDescription() [1/6]

```
smtrat::cadcells::datastructures::CellDescription::Cell↩
Description (
            IndexedRoot bound ) [inline]
```

15.75.2.2 CellDescription() [2/6]

```
smtrat::cadcells::datastructures::CellDescription::Cell↩
Description (
            IndexedRoot lower,
            IndexedRoot upper ) [inline]
```

15.75.2.3 CellDescription() [3/6]

```
smtrat::cadcells::datastructures::CellDescription::Cell↩
Description (
            Bound ,
            IndexedRoot upper ) [inline]
```

15.75.2.4 CellDescription() [4/6]

```
smtrat::cadcells::datastructures::CellDescription::Cell↩
Description (
            IndexedRoot lower,
            Bound ) [inline]
```

15.75.2.5 CellDescription() [5/6]

```
smtrat::cadcells::datastructures::CellDescription::Cell↩
Description (
            Bound ,
            Bound ) [inline]
```

15.75.2.6 CellDescription() [6/6]

```
smtrat::cadcells::datastructures::CellDescription::Cell↩
Description ( ) [inline]
```

15.75.3 Member Function Documentation

15.75.3.1 is_section()

```
bool smtrat::cadcells::datastructures::CellDescription::is_section ( )
const [inline]
```

15.75.3.2 is_sector()

```
bool smtrat::cadcells::datastructures::CellDescription::is_sector ( )
const [inline]
```

15.75.3.3 lower()

```
const auto& smtrat::cadcells::datastructures::CellDescription::lower ( )
const [inline]
```

Returns the lower bound as IndexedRoot if finite or std::nullopt if the lower bound is -oo.

Asserts that the cell is a sector.

15.75.3.4 lower_defining() `std::optional<IndexedRoot> smtrat::cadcells::datastructures::CellDescription::lower_defining ( ) const [inline]`

15.75.3.5 polys() `boost::container::flat_set<PolyRef> smtrat::cadcells::datastructures::CellDescription::polys ( ) const [inline]`

15.75.3.6 section_defining() `const IndexedRoot& smtrat::cadcells::datastructures::CellDescription::section_defining ( ) const [inline]`

In case of a section, the defining indexed root is returned.

15.75.3.7 upper() `const auto& smtrat::cadcells::datastructures::CellDescription::upper ( ) const [inline]`

Returns the upper bound as IndexedRoot if finite or std::nullopt if the upper bound is oo.

Asserts that the cell is a sector.

15.75.3.8 upper_defining() `std::optional<IndexedRoot> smtrat::cadcells::datastructures::CellDescription::upper_defining ( ) const [inline]`

15.76 smtrat::cadcells::datastructures::CellRepresentation< P > Struct Template Reference

Represents a cell.

```
#include <representation.h>
```

Public Member Functions

- CellRepresentation (SampledDerivationRef< P > &deriv)

Data Fields

- CellDescription 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.

15.76.1 Detailed Description

template<typename P>
struct smtrat::cadcells::datastructures::CellRepresentation< P >

Represents a cell.

15.76.2 Constructor & Destructor Documentation

15.76.2.1 CellRepresentation() template<typename P>

```
smtrat::cadcells::datastructures::CellRepresentation< P >::CellRepresentation (
            SampledDerivationRef< P > & deriv )  [inline]
```

15.76.3 Field Documentation

15.76.3.1 derivation template<typename P>

```
SampledDerivationRef<P> smtrat::cadcells::datastructures::CellRepresentation< P >::derivation
```

Derivation.

15.76.3.2 description template<typename P>

```
CellDescription smtrat::cadcells::datastructures::CellRepresentation< P >::description
```

Description of a cell.

15.76.3.3 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.

15.76.3.4 ordering template<typename P>

```
IndexedRootOrdering smtrat::cadcells::datastructures::CellRepresentation< P >::ordering
```

An ordering on the roots that protects the cell.

15.77 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)

15.77.1 Detailed Description

This class takes care of calling the actual solver.

15.77.2 Constructor & Destructor Documentation

15.77.2.1 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

exec	Filename of the solver executable.
timeout	Timeout for a single call to the solver.
original	Filename of the original file to obtain the expected exit code.

15.77.3 Member Function Documentation

15.77.3.1 expectedCode() `int delta::Checker::expectedCode ( ) const [inline]`

Return expected exit code.

Returns Expected exit code.

15.77.3.2 getKilled() `std::size_t delta::Checker::getKilled ( ) const [inline]`

15.77.3.3 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

n	Input data.
temp	Filename of the temporary file.

Returns If the exit code is still the same.

15.77.3.4 resetKilled() `void delta::Checker::resetKilled ( ) const [inline]`

15.77.3.5 resetTimeout() `void delta::Checker::resetTimeout ( std::size_t t ) [inline]`

15.78 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.

15.78.1 Detailed Description

All properties needed to create a submit file.

15.78.2 Field Documentation

15.78.2.1 array_size `std::size_t benchmax::slurm::ChunkedSubmitfileProperties::array_size`

Array size.

15.78.2.2 file_suffix `std::string benchmax::slurm::ChunkedSubmitfileProperties::file_suffix`

Suffix for job and submit file.

15.78.2.3 filename_jobs `std::string benchmax::slurm::ChunkedSubmitfileProperties::filename_jobs`

Filename of the job list file.

15.78.2.4 grace_time `carl::settings::duration benchmax::slurm::ChunkedSubmitfileProperties::grace_time`

Grace time in seconds.

15.78.2.5 job_range `std::pair<std::size_t,std::size_t> benchmax::slurm::ChunkedSubmitfileProperties::job_range`

This slice size.

15.78.2.6 limit_memory `carl::settings::binary_quantity benchmax::slurm::ChunkedSubmitfileProperties::limit_memory`

Memory limit in megabytes.

15.78.2.7 limit_time `carl::settings::duration benchmax::slurm::ChunkedSubmitfileProperties::limit_time`

Time limit in seconds.

15.78.2.8 slice_size `std::size_t benchmax::slurm::ChunkedSubmitfileProperties::slice_size`

Slice size.

15.78.2.9 tmp_dir `std::string benchmax::slurm::ChunkedSubmitfileProperties::tmp_dir`

Temporary directory for log files.

15.79 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 reloced () 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 &ps, bool learnt=false)

15.79.1 Member Function Documentation

15.79.1.1 abstraction()

`uint32_t Minisat::Clause::abstraction ( ) const [inline]`

15.79.1.2 activity() [1/2]

`float& Minisat::Clause::activity ( ) [inline]`

15.79.1.3 activity() [2/2]

`float Minisat::Clause::activity ( ) const [inline]`

15.79.1.4 calcAbstraction()

`void Minisat::Clause::calcAbstraction ( ) [inline]`

15.79.1.5 has_extra()

`bool Minisat::Clause::has_extra ( ) const [inline]`

15.79.1.6 last()

`const Lit& Minisat::Clause::last ( ) const [inline]`

15.79.1.7 learnt()

`bool Minisat::Clause::learnt ( ) const [inline]`

15.79.1.8 mark() [1/2]

`uint32_t Minisat::Clause::mark ( ) const [inline]`

15.79.1.9 mark() [2/2]

`void Minisat::Clause::mark ( uint32_t m ) [inline]`

15.79.1.10 operator const Lit *()

`Minisat::Clause::operator const Lit * ( void ) const [inline]`

15.79.1.11 operator[]() [1/2]

`Lit& Minisat::Clause::operator[] ( int i ) [inline]`

15.79.1.12 operator[]() [2/2]

`Lit Minisat::Clause::operator[] ( int i ) const [inline]`

15.79.1.13 pop()

`void Minisat::Clause::pop ( ) [inline]`

15.79.1.14 relocate()

```
void Minisat::Clause::relocate (
    CRef c ) [inline]
```

15.79.1.15 relocation()

```
CRef Minisat::Clause::relocation ( ) const [inline]
```

15.79.1.16 reloced()

```
bool Minisat::Clause::reloced ( ) const [inline]
```

15.79.1.17 shrink()

```
void Minisat::Clause::shrink (
    int i ) [inline]
```

15.79.1.18 size()

```
int Minisat::Clause::size ( ) const [inline]
```

15.79.1.19 strengthen()

```
void Minisat::Clause::strengthen (
    Lit p ) [inline]
```

15.79.1.20 subsumes()

```
Lit Minisat::Clause::subsumes (
    const Clause & other ) const [inline]
```

15.79.1.21 type()

```
unsigned Minisat::Clause::type ( ) const [inline]
```

15.79.2 Friends And Related Function Documentation

15.79.2.1 Clause_new

```
template<class V >
Clause* Clause_new (
    const V & ps,
    bool learnt = false ) [friend]
```

15.79.2.2 ClauseAllocator

```
friend class ClauseAllocator [friend]
```

15.79.3 Field Documentation

15.79.3.1 abs

```
uint32_t Minisat::Clause::abs
```

15.79.3.2 act

```
float Minisat::Clause::act
```

15.79.3.3 has_extra

```
unsigned Minisat::Clause::has_extra
```

15.79.3.4 lit `Lit Minisat::Clause::lit`

15.79.3.5 mark `unsigned Minisat::Clause::mark`

15.79.3.6 rel `CRef Minisat::Clause::rel`

15.79.3.7 reloced `unsigned Minisat::Clause::reloced`

15.79.3.8 size `unsigned Minisat::Clause::size`

15.79.3.9 type `unsigned Minisat::Clause::type`

15.80 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

15.80.1 Member Typedef Documentation

15.80.1.1 Ref `typedef uint32_t Minisat::RegionAllocator< uint32_t >::Ref [inherited]`

15.80.2 Member Enumeration Documentation

15.80.2.1 anonymous enum `anonymous enum [inherited]`

15.80.2.2 anonymous enum `anonymous enum [inherited]`

15.80.3 Constructor & Destructor Documentation

15.80.3.1 ClauseAllocator() [1/2]

```
Minisat::ClauseAllocator::ClauseAllocator (
    uint32_t start_cap ) [inline]
```

15.80.3.2 ClauseAllocator() [2/2]

```
Minisat::ClauseAllocator::ClauseAllocator ( ) [inline]
```

15.80.4 Member Function Documentation

15.80.4.1 ael() [1/2]

```
Ref Minisat::ClauseAllocator::ael (
    const Clause * t ) [inline]
```

15.80.4.2 ael() [2/2]

```
Ref Minisat::RegionAllocator< uint32_t >::ael (
    const uint32_t * t ) [inline], [inherited]
```

15.80.4.3 alloc() [1/2]

```
template<class Lits >
CRef Minisat::ClauseAllocator::alloc (
    const Lits & ps,
    unsigned _type ) [inline]
```

15.80.4.4 alloc() [2/2]

```
RegionAllocator< uint32_t >::Ref Minisat::RegionAllocator< uint32_t >↩
::alloc (
    int size ) [inherited]
```

15.80.4.5 free() [1/2]

```
void Minisat::ClauseAllocator::free (
    CRef cid ) [inline]
```

15.80.4.6 free() [2/2]

```
void Minisat::RegionAllocator< uint32_t >::free (
    int size ) [inline], [inherited]
```

15.80.4.7 lea() [1/2]

```
Clause* Minisat::ClauseAllocator::lea (
    Ref r ) [inline]
```

15.80.4.8 lea() [2/2]

```
const Clause* Minisat::ClauseAllocator::lea (
    Ref r ) const [inline]
```

15.80.4.9 moveTo() [1/2]

```
void Minisat::ClauseAllocator::moveTo (
    ClauseAllocator & to ) [inline]
```

15.80.4.10 moveTo() [2/2]

```
void Minisat::RegionAllocator< uint32_t >::moveTo (
    RegionAllocator< uint32_t > & to ) [inline], [inherited]
```

15.80.4.11 operator[]() [1/2]

```
Clause& Minisat::ClauseAllocator::operator[] (
    Ref r ) [inline]
```

15.80.4.12 operator[]() [2/2]

```
const Clause& Minisat::ClauseAllocator::operator[] (
    Ref r ) const [inline]
```

15.80.4.13 reloc()

```
void Minisat::ClauseAllocator::reloc (
    CRef & cr,
    ClauseAllocator & to ) [inline]
```

15.80.4.14 size()

```
uint32_t Minisat::RegionAllocator< uint32_t >::size (
    void ) const [inline], [inherited]
```

15.80.4.15 wasted()

```
uint32_t Minisat::RegionAllocator< uint32_t >::wasted ( ) const [inline],
[inherited]
```

15.80.5 Field Documentation

15.80.5.1 extra_clause_field

```
bool Minisat::ClauseAllocator::extra_clause_field
```

15.81 smtrat::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)

15.81.1 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 -> F).

15.81.2 Member Typedef Documentation

15.81.2.1 const_iterator `using smtrat::mcsat::ClauseChain::const_iterator = typename std::vector<Link>↩ ::const_iterator`

15.81.3 Constructor & Destructor Documentation

15.81.3.1 ClauseChain() `smtrat::mcsat::ClauseChain::ClauseChain ( ) [inline]`

15.81.4 Member Function Documentation

15.81.4.1 appendConflicting() `void smtrat::mcsat::ClauseChain::appendConflicting ( const FormulaT && clause ) [inline]`

Append a conflicting clause (regarding the current assignment).

15.81.4.2 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.

15.81.4.3 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 = ∼v for a Tseitin variable v obtained via createTseitinVar.

15.81.4.4 begin()

```
const_iterator smtrat::mcsat::ClauseChain::begin ( ) const [inline]
```

Returns A constant iterator to the beginning of this chain.

15.81.4.5 chain()

```
const std::vector<Link>& smtrat::mcsat::ClauseChain::chain ( ) const [inline]
```

Returns A vector representing this chain.

15.81.4.6 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 <-> formula or at least ∼formula -> ∼C

15.81.4.7 end()

```
const_iterator smtrat::mcsat::ClauseChain::end ( ) const [inline]
```

Returns A constant iterator to the end of this chain.

15.81.4.8 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.

15.81.4.9 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.

15.81.4.10 to_formula() `FormulaT smtrat::mcsat::ClauseChain::to_formula ( ) const`

Transforms the clause chain into a formula (containing Tseitin variables).

15.81.5 Friends And Related Function Documentation

15.81.5.1 operator<< `std::ostream& operator<< ( std::ostream & stream, const ClauseChain & chain ) [friend]`

15.82 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

15.82.1 Member Function Documentation

15.82.1.1 buildModel() `Model smtrat::sat::detail::ClauseChecker::buildModel ( ) const [inline]`

15.82.1.2 operator()() [1/3] `void smtrat::sat::detail::ClauseChecker::operator() ( const FormulasT & formulas ) const [inline]`

15.82.1.3 operator()() [2/3] `void smtrat::sat::detail::ClauseChecker::operator() ( const FormulaT & formula ) const [inline]`

15.82.1.4 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]
```

15.83 smtrat::CMakeOptionPrinter Struct Reference

`#include <compile_information.h>`

Data Fields

- bool advanced

15.83.1 Field Documentation

15.83.1.1 advanced `bool smtrat::CMakeOptionPrinter::advanced`

15.84 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 ()

15.84.1 Member Function Documentation

15.84.1.1 bucket() template<class T >

```
const vec<typename HashTable::Pair>& Minisat::CMap< T >::bucket (
            int i ) const [inline]
```

15.84.1.2 bucket_count() template<class T >

```
int Minisat::CMap< T >::bucket_count ( ) const [inline]
```

15.84.1.3 clear() template<class T >

```
void Minisat::CMap< T >::clear ( ) [inline]
```

15.84.1.4 debug() template<class T >

```
void Minisat::CMap< T >::debug ( ) [inline]
```

15.84.1.5 growTo() template<class T >

```
void Minisat::CMap< T >::growTo (
            CRef cr,
            const T & t ) [inline]
```

15.84.1.6 has() template<class T >

```
bool Minisat::CMap< T >::has (
            CRef cr,
            T & t ) [inline]
```

15.84.1.7 insert() template<class T >

```
void Minisat::CMap< T >::insert (
            CRef cr,
            const T & t ) [inline]
```

15.84.1.8 moveTo() template<class T >

```
void Minisat::CMap< T >::moveTo (
            CMap< T > & other )  [inline]
```

15.84.1.9 operator[]() [1/2] template<class T >

```
T& Minisat::CMap< T >::operator[] (
            CRef cr )  [inline]
```

15.84.1.10 operator[]() [2/2] template<class T >

```
const T& Minisat::CMap< T >::operator[] (
            CRef cr ) const  [inline]
```

15.84.1.11 remove() template<class T >

```
void Minisat::CMap< T >::remove (
            CRef cr )  [inline]
```

15.84.1.12 size() template<class T >

```
int Minisat::CMap< T >::size ( ) const  [inline]
```

15.85 smtrat::CNFerModule Class Reference

```
#include <CNFerModule.h>
```

Data Structures

- struct SettingsType

Public Types

- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::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 smtrat::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 _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.
- 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 &_branching↩ Value) 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 _preferLeft↩ Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩ ::vector< FormulaT >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩ AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩ CaseWeak=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.

15.85.1 Member Enumeration Documentation

15.85.1.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

15.85.2 Constructor & Destructor Documentation

15.85.2.1 CNFerModule()

```
smtrat::CNFerModule::CNFerModule (
    const ModuleInput * _formula,
    Conditionals & _conditionals,
    Manager * const _manager = NULL )
```

Constructs a CNFerModule.

15.85.2.2 ~CNFerModule() `smtrat::CNFerModule::~CNFerModule ( )`

Destructs a CNFerModule.

15.85.3 Member Function Documentation

15.85.3.1 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.

15.85.3.2 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 >.

15.85.3.3 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 smtrat::SATModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::VSModule< Settings >, smtrat::PFEModule< Settings >, smtrat::EQModule< Setting smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Sett

smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, and smtrat::GBModule< Settings >.

15.85.3.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.85.3.5 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.

15.85.3.6 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.

15.85.3.7 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.

15.85.3.8 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.

15.85.3.9 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

_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.

15.85.3.10 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

_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.

15.85.3.11 allModels() `const std::vector<Model>& smtrat::Module::allModels ( ) const [inline], [inherited]`

Returns All satisfying assignments, if existent.

15.85.3.12 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.

15.85.3.13 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.

15.85.3.14 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.

15.85.3.15 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.

15.85.3.16 branchAt() [1/6]

```
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]
```

15.85.3.17 branchAt() [2/6]

```
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]
```

15.85.3.18 branchAt() [3/6]

```
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]
```

15.85.3.19 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.85.3.20 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            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]
```

15.85.3.21 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

15.85.3.22 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

15.85.3.23 checkCore() `Answer smtrat::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 smtrat::Module.

15.85.3.24 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
    std::vector< FormulaSetT >::const_iterator _infsubset,
    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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.85.3.25 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.

15.85.3.26 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.85.3.27 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.85.3.28 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.85.3.29 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.85.3.30 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.85.3.31 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

15.85.3.32 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.

15.85.3.33 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.85.3.34 constraintsToInform() `const carl::FastSet<FormulaT>& smtrat::Module::constraintsToInform ( ) const [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.85.3.35 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

15.85.3.36 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)

15.85.3.37 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

_constraint	The constraint to remove from internal data structures.

15.85.3.38 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.85.3.39 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.85.3.40 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.85.3.41 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.85.3.42 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
            const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.85.3.43 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.

15.85.3.44 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.

15.85.3.45 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

15.85.3.46 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

15.85.3.47 getBackendsAllModels()

`void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

15.85.3.48 getBackendsModel()

`void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

15.85.3.49 getInfeasibleSubsets() [1/2]

`void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

15.85.3.50 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.85.3.51 getInformationRelevantFormulas() `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.85.3.52 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.

15.85.3.53 getOrigins() [1/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

15.85.3.54 getOrigins() [2/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

15.85.3.55 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.85.3.56 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 reveived formula, if the Boolean is true.

Reimplemented from smtrat::Module.

15.85.3.57 hasLemmas()

`bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

15.85.3.58 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).

15.85.3.59 id()

`std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns A unique ID to identify this module instance.

15.85.3.60 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.

15.85.3.61 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.85.3.62 informBackends()

```
void smtrat::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.

15.85.3.63 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, and smtrat::ICPModule< ICPSettings1 >.

15.85.3.64 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.85.3.65 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::FPPModule< Settings >.

15.85.3.66 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.85.3.67 isLemmaLevel() `bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.85.3.68 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().

15.85.3.69 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.

15.85.3.70 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.85.3.71 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.85.3.72 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.

15.85.3.73 moduleName() `virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]`

Returns The name of the given module type as name.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::EQPreprocessingModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::VSModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::GBModule< Settings >, smtrat::IntEqModule< Setti smtrat::CubeLIAModule< Settings >, smtrat::CurryModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IncWidthModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::BEModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::SplitSOSModule< Settings >.

15.85.3.74 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

15.85.3.75 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

15.85.3.76 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula. TODO: disable this method

15.85.3.77 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula.

15.85.3.78 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to the backends of this module.

15.85.3.79 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to this module.

15.85.3.80 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.85.3.81 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

_out	The stream to print the assignment on.

15.85.3.82 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.85.3.83 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

_out	The stream to print the assignment on.

15.85.3.84 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.85.3.85 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.85.3.86 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.

15.85.3.87 receivedFormulaChecked()

```
void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]
```

Notifies that the received formulas has been checked.

15.85.3.88 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAsInfeasible↩Subset (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

15.85.3.89 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const [inline], [inherited]
```

15.85.3.90 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

_subformula	The sub formula of the received formula to remove.

Reimplemented from smtrat::Module.

15.85.3.91 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

formula	The sub formula of the received formula to remove.

Reimplemented in smtrat::SATModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::VSModule< Settings >, smtrat::PFEModule< Settings >, smtrat::EQModule< Setting smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Sett smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, and smtrat::GBModule< Settings >.

15.85.3.92 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.85.3.93 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins
(
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

15.85.3.94 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline],
[inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.85.3.95 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.85.3.96 runBackends() [1/2]

```
virtual Answer smtrat::PModule::runBackends ( ) [inline],
[virtual], [inherited]
```

Reimplemented from smtrat::Module.

15.85.3.97 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.

15.85.3.98 setId()

```
void smtrat::Module::setId (
    std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.85.3.99 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.

15.85.3.100 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.

15.85.3.101 splitUnequalConstraint()

```
void smtrat::Module::splitUnequalConstraint (
    const FormulaT & _unequalConstraint ) [protected], [inherited]
```

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.85.3.102 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.85.3.103 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 >.

15.85.3.104 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.85.3.105 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.

15.85.3.106 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).

15.85.4 Field Documentation

15.85.4.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.85.4.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.85.4.3 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.

15.85.4.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.85.4.5 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

15.85.4.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.85.4.7 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.

15.85.4.8 mFirstUncheckedReceivedSubformula `ModuleInput::const_iterator smtrat::Module::m↩ FirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

15.85.4.9 mFoundAnswer `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.85.4.10 mFullCheck `bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.85.4.11 mInfeasibleSubsets `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.85.4.12 mInformedConstraints `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.85.4.13 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.

15.85.4.14 mLemmas `std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.85.4.15 mModel `Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.85.4.16 mModelComputed `bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.85.4.17 mNumOfBranchVarsToStore `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.85.4.18 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.

15.85.4.19 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]`

Reusable splitting variables.

15.85.4.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.85.4.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.85.4.22 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.

15.85.4.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

15.85.4.24 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).

15.85.4.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.86 smtrat::CoCoAGBModule< Settings > Class Template Reference

`#include <CoCoAGBModule.h>`

Public Types

- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::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 smtrat::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 _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.

- 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
- 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 &_branching↩Value) 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 _preferLeft↩Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩::vector< FormulaT >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩CaseWeak=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.

15.86.1 Member Enumeration Documentation

15.86.1.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

15.86.2 Member Function Documentation

15.86.2.1 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

_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.

15.86.2.2 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

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

15.86.2.3 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

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 smtrat::SATModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::VSModule< Settings >, smtrat::PFEModule< Settings >, smtrat::EQModule< Setting smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >,

smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Sett smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, and smtrat::GBModule< Settings >.

15.86.2.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.86.2.5 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.

15.86.2.6 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.

15.86.2.7 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.

15.86.2.8 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.

15.86.2.9 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.

15.86.2.10 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

_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.

15.86.2.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns All satisfying assignments, if existent.

15.86.2.12 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.

15.86.2.13 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.

15.86.2.14 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.

15.86.2.15 branchAt() [1/6]

```
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]
```

15.86.2.16 branchAt() [2/6]

```
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]
```

15.86.2.17 branchAt() [3/6]

```
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]
```

15.86.2.18 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.86.2.19 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            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]
```

15.86.2.20 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

15.86.2.21 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in smtrat::PModule.

15.86.2.22 checkCore() `Answer smtrat::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 smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::SATModule< Settings >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRAS smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::EQPreprocessingModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::VSModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::GBModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, smtrat::LVEModule< Settings >, smtrat::ESModule< Settings >, smtrat::GBPPModule< Settings >, smtrat::CNFerModule, smtrat::BEModule< Settings >, smtrat::MCBModule< Settings >, smtrat::PFEModule< Settings >, smtrat::SplitSOSModule< Setti smtrat::EMModule< Settings >, and smtrat::SymmetryModule< Settings >.

15.86.2.23 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
    std::vector< FormulaSetT >::const_iterator _infsubset,
    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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.86.2.24 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.

15.86.2.25 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.86.2.26 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.86.2.27 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.86.2.28 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.86.2.29 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.86.2.30 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

15.86.2.31 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

_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.

15.86.2.32 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.86.2.33 constraintsToInform()

```
const carl::FastSet<FormulaT>& smtrat::Module::constraintsTo↩
Inform ( ) const [inline], [inherited]
```

Returns The constraints which the backends must be informed about.

15.86.2.34 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

15.86.2.35 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)

15.86.2.36 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

_constraint	The constraint to remove from internal data structures.

15.86.2.37 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.86.2.38 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.86.2.39 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.86.2.40 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceived↩
VariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.86.2.41 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBest↩
Origin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.86.2.42 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.

15.86.2.43 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.

15.86.2.44 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

15.86.2.45 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

15.86.2.46 getBackendsAllModels()

`void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

15.86.2.47 getBackendsModel()

`void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

15.86.2.48 getInfeasibleSubsets() [1/2]

`void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

15.86.2.49 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.86.2.50 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.86.2.51 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.

15.86.2.52 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.86.2.53 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.86.2.54 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.86.2.55 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 reveived formula, if the Boolean is true.

Reimplemented in smtrat::PModule.

15.86.2.56 hasLemmas()

```
bool smtrat::Module::hasLemmas ( ) [inline], [inherited]
```

Checks whether this module or any of its backends provides any lemmas.

15.86.2.57 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).

15.86.2.58 id()

```
std::size_t smtrat::Module::id ( ) const [inline], [inherited]
```

Returns A unique ID to identify this module instance.

15.86.2.59 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.

15.86.2.60 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.86.2.61 informBackends()

```
void smtrat::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.

15.86.2.62 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, and smtrat::ICPModule< ICPSettings1 >.

15.86.2.63 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.86.2.64 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::FPPModule< Settings >.

15.86.2.65 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.86.2.66 isLemmaLevel() `bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.86.2.67 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().

15.86.2.68 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.

15.86.2.69 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.86.2.70 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.86.2.71 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

_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.

15.86.2.72 moduleName() `virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]`

Returns The name of the given module type as name.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::EQPreprocessingModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::VSModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::GBModule< Settings >, smtrat::IntEqModule< Setti smtrat::CubeLIAModule< Settings >, smtrat::CurryModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IncWidthModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::BEModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::SplitSOSModule< Settings >.

15.86.2.73 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

15.86.2.74 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

15.86.2.75 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula. TODO: disable this method

15.86.2.76 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula.

15.86.2.77 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to the backends of this module.

15.86.2.78 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to this module.

15.86.2.79 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.86.2.80 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

_out	The stream to print the assignment on.

15.86.2.81 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.86.2.82 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

_out	The stream to print the assignment on.

15.86.2.83 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.86.2.84 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.86.2.85 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.

15.86.2.86 receivedFormulaChecked()

```
void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]
```

Notifies that the received formulas has been checked.

15.86.2.87 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAsInfeasible↩Subset (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

15.86.2.88 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const [inline], [inherited]
```

15.86.2.89 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

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

15.86.2.90 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

formula	The sub formula of the received formula to remove.

Reimplemented in smtrat::SATModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::VSModule< Settings >, smtrat::PFEModule< Settings >, smtrat::EQModule< Setting smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Sett smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, and smtrat::GBModule< Settings >.

15.86.2.91 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.86.2.92 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins
(
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

15.86.2.93 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline],
[inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.86.2.94 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.86.2.95 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( ) [inline], [protected],
[virtual], [inherited]
```

Reimplemented in smtrat::PModule.

15.86.2.96 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

_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 in smtrat::PModule.

15.86.2.97 setId()

```
void smtrat::Module::setId (
    std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.86.2.98 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.

15.86.2.99 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.

15.86.2.100 splitUnequalConstraint()

```
void smtrat::Module::splitUnequalConstraint (
    const FormulaT & _unequalConstraint ) [protected], [inherited]
```

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.86.2.101 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.86.2.102 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 >.

15.86.2.103 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.86.2.104 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::SATModule< Settings >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRAS smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::EQPreprocessingModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::PModule, smtrat::VSModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, smtrat::LVEModule< Settings >, smtrat::MCBModule< Settings >, smtrat::ESModule< Settings >, and smtrat::GBPPModule< Settings >.

15.86.2.105 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).

15.86.3 Field Documentation

15.86.3.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.86.3.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.86.3.3 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.

15.86.3.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.86.3.5 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

15.86.3.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.86.3.7 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.

15.86.3.8 mFirstUncheckedReceivedSubformula `ModuleInput::const_iterator smtrat::Module::m↩FirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

15.86.3.9 mFoundAnswer `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.86.3.10 mFullCheck `bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.86.3.11 mInfeasibleSubsets `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.86.3.12 mInformedConstraints `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.86.3.13 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.

15.86.3.14 mLemmas `std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.86.3.15 mModel `Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.86.3.16 mModelComputed `bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.86.3.17 mNumOfBranchVarsToStore `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.86.3.18 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.

15.86.3.19 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]`

Reusable splitting variables.

15.86.3.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.86.3.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.86.3.22 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.

15.86.3.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

15.86.3.24 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).

15.86.3.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.87 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

15.87.1 Constructor & Destructor Documentation

15.87.1.1 CollectBoolsInUEQs() [1/3]

```
smtrat::CollectBoolsInUEQs::CollectBoolsInUEQs (
    const CollectBoolsInUEQs & ) [delete]
```

15.87.1.2 CollectBoolsInUEQs() [2/3]

```
smtrat::CollectBoolsInUEQs::CollectBoolsInUEQs ( ) [default]
```

15.87.1.3 CollectBoolsInUEQs() [3/3]

```
smtrat::CollectBoolsInUEQs::CollectBoolsInUEQs (
    CollectBoolsInUEQs && ) [default]
```

15.87.2 Member Function Documentation

15.87.2.1 foundBoolInUEQ()

```
bool smtrat::CollectBoolsInUEQs::foundBoolInUEQ ( ) [inline]
```

15.87.2.2 operator=() [1/2]

```
CollectBoolsInUEQs& smtrat::CollectBoolsInUEQs::operator= (
    CollectBoolsInUEQs && ) [default]
```

15.87.2.3 operator=() [2/2]

```
CollectBoolsInUEQs& smtrat::CollectBoolsInUEQs::operator= (
    const CollectBoolsInUEQs & ) [delete]
```

15.87.2.4 visit_ueq()

```
void smtrat::CollectBoolsInUEQs::visit_ueq (
    const FormulaT & formula ) [inline]
```

15.87.3 Friends And Related Function Documentation

15.87.3.1 BoolUEQRewriter

```
friend struct BoolUEQRewriter [friend]
```

15.88 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 ()

15.88.1 Member Typedef Documentation

15.88.1.1 const_iterator

```
template<typename Element, typename Origin>
typedef Super::const_iterator smtrat::CollectionWithOrigins< Element, Origin >::const_iterator
```

15.88.1.2 iterator

```
template<typename Element, typename Origin>
typedef Super::iterator smtrat::CollectionWithOrigins< Element, Origin >::iterator
```

15.88.2 Constructor & Destructor Documentation

15.88.2.1 CollectionWithOrigins()

```
template<typename Element, typename Origin>
smtrat::CollectionWithOrigins< Element, Origin >::CollectionWithOrigins (
            bool _trackElementsWithoutOrigins = true )  [inline]
```

15.88.3 Member Function Documentation

15.88.3.1 add()

```
template<typename Element, typename Origin>
bool smtrat::CollectionWithOrigins< Element, Origin >::add (
            const Element & _element,
            const Origin & _origin )  [inline]
```

15.88.3.2 begin()

```
template<typename Element, typename Origin>
iterator smtrat::CollectionWithOrigins< Element, Origin >::begin ( )  [inline]
```

15.88.3.3 cbegin()

```
template<typename Element, typename Origin>
const_iterator smtrat::CollectionWithOrigins< Element, Origin >::cbegin ( ) const  [inline]
```

15.88.3.4 cend()

```
template<typename Element, typename Origin>
const_iterator smtrat::CollectionWithOrigins< Element, Origin >::cend ( ) const  [inline]
```

15.88.3.5 clearElementsWithoutOrigins()

```
template<typename Element, typename Origin>
void smtrat::CollectionWithOrigins< Element, Origin >::clearElementsWithoutOrigins ( )  [inline]
```

15.88.3.6 contains()

```
template<typename Element, typename Origin>
bool smtrat::CollectionWithOrigins< Element, Origin >::contains (
            const Element & _element )  [inline]
```

15.88.3.7 elementsWithoutOrigins()

```
template<typename Element, typename Origin>
const std::set<Element>& smtrat::CollectionWithOrigins< Element, Origin >::elementsWithout↩
Origins ( ) const  [inline]
```

15.88.3.8 end()

```
template<typename Element, typename Origin>
iterator smtrat::CollectionWithOrigins< Element, Origin >::end ( )  [inline]
```

15.88.3.9 removeOrigin()

```
template<typename Element, typename Origin>
bool smtrat::CollectionWithOrigins< Element, Origin >::removeOrigin (
            const Origin & _origin )  [inline]
```

15.88.3.10 removeOrigins()

```
template<typename Element, typename Origin>
bool smtrat::CollectionWithOrigins< Element, Origin >::removeOrigins (
            const std::set< Origin > & _origins )  [inline]
```

15.89 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

15.89.1 Detailed Description

template<class Settings>
struct smtrat::ICPModule< Settings >::comp

Typedefs:

15.89.2 Member Function Documentation

15.89.2.1 operator()()

```
template<class Settings>
bool smtrat::ICPModule< Settings >::comp::operator() (
            std::pair< double, unsigned > lhs,
            std::pair< double, unsigned > rhs ) const  [inline]
```

15.90 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 getId () 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)

15.90.1 Constructor & Destructor Documentation

15.90.1.1 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 )
```

15.90.1.2 Condition() [2/3]

```
smtrat::vs::Condition::Condition (
    const Condition & _cond,
    size_t _id )
```

15.90.1.3 Condition() [3/3]

```
smtrat::vs::Condition::Condition (
    const Condition & ) [delete]
```

15.90.1.4 ~Condition()

```
smtrat::vs::Condition::~Condition ( )
```

15.90.2 Member Function Documentation

15.90.2.1 constraint()

```
const smtrat::ConstraintT& smtrat::vs::Condition::constraint ( ) const [inline]
```

15.90.2.2 flag()

```
bool smtrat::vs::Condition::flag ( ) const [inline]
```

15.90.2.3 getId()

```
size_t smtrat::vs::Condition::getId ( ) const [inline]
```

15.90.2.4 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

_condition	The condition to compare with (rhs).

Returns true ,if the given substitution is greater than this substitution; false ,otherwise.

15.90.2.5 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

_condition	The condition to compare with (rhs).

Returns true ,if the given condition is equal to this condition; false ,otherwise.

15.90.2.6 originalConditions()

```
const carl::PointerSet<Condition>& smtrat::vs::Condition::original↩
Conditions ( ) const [inline]
```

15.90.2.7 pOriginalConditions()

```
carl::PointerSet<Condition>* smtrat::vs::Condition::pOriginal↩
Conditions ( ) const [inline]
```

15.90.2.8 print()

```
void smtrat::vs::Condition::print (
    std::ostream & _out = std::cout ) const
```

Prints the condition to an output stream.

Parameters

_out	The output stream, where it should print.

15.90.2.9 recentlyAdded()

```
bool smtrat::vs::Condition::recentlyAdded ( ) const [inline]
```

15.90.2.10 rFlag()

```
bool& smtrat::vs::Condition::rFlag ( ) const [inline]
```

15.90.2.11 rRecentlyAdded()

```
bool& smtrat::vs::Condition::rRecentlyAdded ( ) const [inline]
```

15.90.2.12 rValuation()

```
size_t& smtrat::vs::Condition::rValuation ( ) const [inline]
```

15.90.2.13 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

_consideredVariable	The variable which is considered in this valuation.
_maxNumberOfVars	

Returns A valuation of the constraint according to an heuristic.

15.90.2.14 valuation() `size_t smtrat::vs::Condition::valuation ( ) const [inline]`

15.90.3 Friends And Related Function Documentation

15.90.3.1 operator<< `std::ostream& operator<< ( std::ostream & _out, const Condition & _condition ) [friend]`

15.91 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.

15.91.1 Detailed Description

Backend for the HTCondor batch system.

Currently submits all jobs individually and asynchronously waits for them to finish.

15.91.2 Member Function Documentation

15.91.2.1 addResult()

```
void benchmax::Backend::addResult (
    const Tool * tool,
    const fs::path & file,
    BenchmarkResult && result ) [inline], [inherited]
```

Add a result.

15.91.2.2 collect_results()

```
virtual bool benchmax::Backend::collect_results (
    const Jobs & ,
    bool ) [inline], [protected], [virtual], [inherited]
```

15.91.2.3 execute()

```
virtual void benchmax::CondorBackend::execute (
    const Tool * ,
    const fs::path & ) [inline], [protected], [virtual]
```

No-op version of execute.

Reimplemented from benchmax::Backend.

15.91.2.4 finalize()

```
virtual void benchmax::Backend::finalize ( ) [inline], [protected], [virtual],
[inherited]
```

Hook to allow for asynchronous backends to wait for jobs to terminate.

15.91.2.5 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.

15.91.2.6 process_results()

```
void benchmax::Backend::process_results (
    const Jobs & jobs,
    bool check_finished ) [inline], [inherited]
```

15.91.2.7 run()

```
void benchmax::CondorBackend::run (
    const Jobs & ,
    bool ) [inline]
```

Run all tools on all benchmarks.

15.91.2.8 sanitize_results()

```
void benchmax::Backend::sanitize_results (
    const Jobs & jobs ) const [inline], [protected], [inherited]
```

15.91.2.9 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.

15.91.2.10 suspendable()

```
bool benchmax::Backend::suspendable ( ) const [inline], [inherited]
```

15.91.2.11 write_results()

```
void benchmax::Backend::write_results (
    const Jobs & jobs ) const [inline], [protected], [inherited]
```

15.91.3 Field Documentation

15.91.3.1 mExpectedJobs

```
std::size_t benchmax::Backend::mExpectedJobs [protected], [inherited]
```

Number of jobs that should be run.

15.91.3.2 mFinishedJobs

```
std::atomic<std::size_t> benchmax::Backend::mFinishedJobs [protected],
[inherited]
```

Number of jobs that are finished.

15.91.3.3 mLastPercent

```
std::atomic<std::size_t> benchmax::Backend::mLastPercent [protected],
[inherited]
```

Percentage of finished jobs when madeProgress() was last called.

15.92 smtrat::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)

15.92.1 Constructor & Destructor Documentation

15.92.1.1 ConflictGenerator()

```
template<class Comparator >
smtrat::mcsat::fm::ConflictGenerator< Comparator >::ConflictGenerator (
    const std::vector< ConstraintT > & bounds,
    const Model & m,
    carl::Variable v ) [inline]
```

15.92.2 Member Function Documentation

15.92.2.1 generateExplanation()

```
template<class Comparator >
template<typename Callback >
void smtrat::mcsat::fm::ConflictGenerator< Comparator >::generateExplanation (
            Callback && callback ) [inline]
```

15.93 smtrat::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)
- std::size_t coveredSamples (std::size_t id) const
- void addSample (const Sample &sample)
- std::size_t getMaxDegreeConstraint () const

  Retrieves the constraint that covers the most samples.
- void selectConstraint (std::size_t id)

  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)

15.93.1 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.

15.93.2 Constructor & Destructor Documentation

15.93.2.1 ConflictGraph()

```
smtrat::cad::ConflictGraph::ConflictGraph (
            std::size_t constraints ) [inline]
```

15.93.3 Member Function Documentation

15.93.3.1 addSample()

```
void smtrat::cad::ConflictGraph::addSample (
            const Sample & sample )
```

15.93.3.2 coveredSamples()

```
std::size_t smtrat::cad::ConflictGraph::coveredSamples (
            std::size_t id ) const [inline]
```

15.93.3.3 getData()

```
carl::Bitset smtrat::cad::ConflictGraph::getData (
    std::size_t id )
```

15.93.3.4 getMaxDegreeConstraint()

```
std::size_t smtrat::cad::ConflictGraph::getMaxDegreeConstraint
( ) const
```

Retrieves the constraint that covers the most samples.

15.93.3.5 getRemainingConstraints()

```
std::vector< std::pair< std::size_t, carl::Bitset > >
smtrat::cad::ConflictGraph::getRemainingConstraints ( )
```

15.93.3.6 hasRemainingSamples()

```
bool smtrat::cad::ConflictGraph::hasRemainingSamples ( )
const
```

Checks if there are samples still uncovered.

15.93.3.7 numRemainingConstraints()

```
std::size_t smtrat::cad::ConflictGraph::numRemaining↩
Constraints ( ) const
```

15.93.3.8 numSamples()

```
std::size_t smtrat::cad::ConflictGraph::numSamples ( ) const
```

15.93.3.9 removeDuplicateColumns()

```
ConflictGraph smtrat::cad::ConflictGraph::removeDuplicate↩
Columns ( )
```

Returns a new ConflictGraph whose adjacency matrix consists only of the unique columns of the adjacency matrix of this graph.

15.93.3.10 selectConstraint()

```
void smtrat::cad::ConflictGraph::selectConstraint (
    std::size_t id )
```

Removes the given constraint and disable all sample points covered by this constraint.

15.93.3.11 selectEssentialConstraints()

```
std::vector< std::size_t > smtrat::cad::ConflictGraph↩
::selectEssentialConstraints ( )
```

Selects all essential constraints and returns their indices.

15.93.4 Friends And Related Function Documentation

15.93.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const ConflictGraph & cg ) [friend]
```

15.94 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

15.94.1 Detailed Description

Helper functor for addConstants() method.

15.94.2 Constructor & Destructor Documentation

15.94.2.1 ConstantAdder()

```
smtrat::parser::Theories::ConstantAdder::ConstantAdder (
    qi::symbols< char, types::ConstType > & constants ) [inline]
```

15.94.3 Member Function Documentation

15.94.3.1 operator()()

```
template<typename T >
void smtrat::parser::Theories::ConstantAdder::operator() (
    T * ) [inline]
```

15.94.4 Field Documentation

15.94.4.1 constants

```
qi::symbols<char, types::ConstType>& smtrat::parser::Theories::Constant↩
Adder::constants
```

15.95 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

15.95.1 Member Function Documentation

15.95.1.1 complexity()

```
template<Backtracking BT>
std::size_t smtrat::cad::CADConstraints< BT >::ConstraintComparator::complexity (
    const ConstraintT & c ) const [inline]
```

15.95.1.2 operator()()

```
template<Backtracking BT>
bool smtrat::cad::CADConstraints< BT >::ConstraintComparator::operator() (
    const ConstraintT & lhs,
    const ConstraintT & rhs ) const [inline]
```

15.96 smtrat::cad::preprocessor::ConstraintUpdate Struct Reference

```
#include <CADPreprocessor.h>
```

Data Fields

- std::vector< ConstraintT > toAdd
- std::vector< ConstraintT > toRemove

15.96.1 Field Documentation

15.96.1.1 toAdd `std::vector< ConstraintT > smtrat::cad::preprocessor::ConstraintUpdate::toAdd`

15.96.1.2 toRemove `std::vector< ConstraintT > smtrat::cad::preprocessor::ConstraintUpdate↩ ::toRemove`

15.97 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

15.97.1 Constructor & Destructor Documentation

15.97.1.1 ConstrTree() `smtrat::ConstrTree::ConstrTree ( const ConstraintT & _constraint ) [inline]`

15.97.1.2 ~ConstrTree() `smtrat::ConstrTree::~ConstrTree ( ) [inline]`

15.97.2 Member Function Documentation

15.97.2.1 constraint() `const ConstraintT& smtrat::ConstrTree::constraint ( ) const [inline]`

15.97.2.2 left() `const PolyTree& smtrat::ConstrTree::left ( ) const [inline]`

15.97.2.3 relation() `carl::Relation smtrat::ConstrTree::relation ( ) const [inline]`

15.97.2.4 right() `const PolyTree& smtrat::ConstrTree::right ( ) const [inline]`

15.98 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.

15.98.1 Detailed Description

This class takes care of asynchronous execution of calls to the solver.

15.98.2 Constructor & Destructor Documentation

15.98.2.1 Consumer()

```
delta::Consumer::Consumer (
    const std::string & tempPrefix,
    std::size_t threads,
    const Checker & checker ) [inline]
```

Constructor.

Parameters

tempPrefix	Prefix for temporary files.

15.98.2.2 ~Consumer()

```
delta::Consumer::~Consumer ( ) [inline]
```

Destructor.

15.98.3 Member Function Documentation

15.98.3.1 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

n	Node to check.
message	Message.
num	Number of this node in the iteration.

15.98.3.2 getProgress() `std::pair<unsigned, unsigned> delta::Consumer::getProgress ( ) const [inline]`

Returns the current progress.

Returns Pair of finished and total checks.

15.98.3.3 getResult() `std::tuple<Node, std::string, std::size_t> delta::Consumer::getResult ( ) [inline]`

Returns the result, assuming that one exists.

Returns Result.

15.98.3.4 hasResult() `bool delta::Consumer::hasResult ( ) const [inline]`

Checks if at least one job was successful.

Returns If a result is there.

15.98.3.5 reset() `void delta::Consumer::reset ( ) [inline]`

Reset this executor.

Waits for all jobs to finish and resets the internal status.

15.98.3.6 wait() `bool delta::Consumer::wait ( ) [inline]`

Wait for at least one job to finish.

Returns If all jobs have finished.

15.99 smtrat::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.

15.99.1 Detailed Description

template<class Settings>
struct smtrat::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.

TODO: Maybe it is enough to store a mapping of the formula to the position.

15.99.2 Constructor & Destructor Documentation

15.99.2.1 Context() template<class Settings>

```
smtrat::LRAModule< Settings >::Context::Context (
            const FormulaT & _origin,
            ModuleInput::iterator _pos )  [inline]
```

15.99.3 Field Documentation

15.99.3.1 origin template<class Settings>

```
FormulaT smtrat::LRAModule< Settings >::Context::origin
```

The formula in the received formula.

15.99.3.2 position template<class Settings>

```
ModuleInput::iterator smtrat::LRAModule< Settings >::Context::position
```

The position of this formula in the passed formula.

15.100 smtrat::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 isLinear () 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

15.100.1 Constructor & Destructor Documentation

15.100.1.1 ContractionCandidate() [1/4]

```
smtrat::icp::ContractionCandidate::ContractionCandidate (
    const ContractionCandidate & _original ) [delete]
```

Constructors:

15.100.1.2 ContractionCandidate() [2/4] smtrat::icp::ContractionCandidate::ContractionCandidate ( ) [delete]

15.100.1.3 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]
```

15.100.1.4 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

_constraint	
_derivationVar	

15.100.1.5 ~ContractionCandidate() smtrat::icp::ContractionCandidate::~ContractionCandidate ( ) [inline]

Destructor:

15.100.2 Member Function Documentation

15.100.2.1 activity() size_t smtrat::icp::ContractionCandidate::activity ( ) [inline]

15.100.2.2 addICPVariable() void smtrat::icp::ContractionCandidate::addICPVariable (

```
IcpVariable * _icpVar ) [inline]
```

15.100.2.3 addOrigin() void smtrat::icp::ContractionCandidate::addOrigin (

```
const FormulaT & _origin )
```

15.100.2.4 calcDerivative()

```
void smtrat::icp::ContractionCandidate::calcDerivative ( ) throw ( ) [inline]
```

15.100.2.5 calcRWA()

```
double smtrat::icp::ContractionCandidate::calcRWA ( ) [inline]
```

15.100.2.6 constraint()

```
const ConstraintT& smtrat::icp::ContractionCandidate::constraint ( ) const [inline]
```

15.100.2.7 contract()

```
bool smtrat::icp::ContractionCandidate::contract (
    EvalDoubleIntervalMap & _intervals,
    DoubleInterval & _resA,
    DoubleInterval & _resB ) [inline]
```

15.100.2.8 contractor()

```
Contractor<carl::SimpleNewton>& smtrat::icp::ContractionCandidate↩
::contractor ( ) const [inline]
```

15.100.2.9 derivationVar()

```
carl::Variable::Arg smtrat::icp::ContractionCandidate::derivationVar
( ) const [inline]
```

15.100.2.10 derivative()

```
const Poly& smtrat::icp::ContractionCandidate::derivative ( ) const
[inline]
```

15.100.2.11 hasOrigin()

```
bool smtrat::icp::ContractionCandidate::hasOrigin (
    const FormulaT & _origin ) const [inline]
```

15.100.2.12 id()

```
unsigned smtrat::icp::ContractionCandidate::id ( ) const [inline]
```

15.100.2.13 incrementReusagesAfterTargetDiameterReached()

```
unsigned smtrat::icp::Contraction↩
Candidate::incrementReusagesAfterTargetDiameterReached ( ) [inline]
```

15.100.2.14 isActive()

```
bool smtrat::icp::ContractionCandidate::isActive ( ) const [inline]
```

15.100.2.15 isDerived()

```
bool smtrat::icp::ContractionCandidate::isDerived ( ) const [inline]
```

15.100.2.16 isLinear() `bool smtrat::icp::ContractionCandidate::isLinear ( ) const [inline]`

15.100.2.17 lastPayoff() `double smtrat::icp::ContractionCandidate::lastPayoff ( ) const [inline]`

15.100.2.18 lastRWA() `double smtrat::icp::ContractionCandidate::lastRWA ( ) const [inline]`

15.100.2.19 lhs() `carl::Variable::Arg smtrat::icp::ContractionCandidate::lhs ( ) const [inline]`

15.100.2.20 operator<() `bool smtrat::icp::ContractionCandidate::operator< ( ContractionCandidate const & rhs ) const [inline]`

15.100.2.21 operator==() `bool smtrat::icp::ContractionCandidate::operator== ( ContractionCandidate const & rhs ) const [inline]`

15.100.2.22 origin() `const FormulaSetT& smtrat::icp::ContractionCandidate::origin ( ) const [inline]`

15.100.2.23 print() `void smtrat::icp::ContractionCandidate::print ( std::ostream & _out = std::cout ) const [inline]`

15.100.2.24 removeOrigin() `void smtrat::icp::ContractionCandidate::removeOrigin ( const FormulaT & _origin )`

15.100.2.25 resetReusagesAfterTargetDiameterReached() `void smtrat::icp::ContractionCandidate↩ ::resetReusagesAfterTargetDiameterReached ( ) [inline]`

15.100.2.26 resetWeights() `void smtrat::icp::ContractionCandidate::resetWeights ( ) [inline]`

15.100.2.27 reusagesAfterTargetDiameterReached() `unsigned smtrat::icp::ContractionCandidate↩ ::reusagesAfterTargetDiameterReached ( ) const [inline]`

15.100.2.28 rhs() `const Poly& smtrat::icp::ContractionCandidate::rhs ( ) const [inline]`

Functions:

15.100.2.29 rOrigin()

`FormulaSetT& smtrat::icp::ContractionCandidate::rOrigin ( ) [inline]`

15.100.2.30 RWA()

`double smtrat::icp::ContractionCandidate::RWA ( ) const [inline]`

15.100.2.31 setDerivationVar()

`void smtrat::icp::ContractionCandidate::setDerivationVar ( carl::Variable _var ) [inline]`

15.100.2.32 setLhs()

`void smtrat::icp::ContractionCandidate::setLhs ( carl::Variable _lhs ) [inline]`

15.100.2.33 setLinear()

`void smtrat::icp::ContractionCandidate::setLinear ( ) [inline]`

15.100.2.34 setNonlinear()

`void smtrat::icp::ContractionCandidate::setNonlinear ( ) [inline]`

15.100.2.35 setPayoff()

`void smtrat::icp::ContractionCandidate::setPayoff ( double _weight ) [inline]`

15.100.2.36 updateLastRWA()

`void smtrat::icp::ContractionCandidate::updateLastRWA ( ) [inline]`

15.101 smtrat::icp::contractionCandidateComp Struct Reference

`#include <ContractionCandidate.h>`

Public Member Functions

- bool operator() (const ContractionCandidate *const lhs, const ContractionCandidate *const rhs) const

15.101.1 Member Function Documentation

15.101.1.1 operator()()

```
bool smtrat::icp::contractionCandidateComp::operator() (
    const ContractionCandidate *const lhs,
    const ContractionCandidate *const rhs ) const [inline]
```

15.102 smtrat::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 ()

15.102.1 Constructor & Destructor Documentation

15.102.1.1 ContractionCandidateManager() `smtrat::icp::ContractionCandidateManager::Contraction↩ CandidateManager ( )`

Constructors.

15.102.1.2 ~ContractionCandidateManager() `smtrat::icp::ContractionCandidateManager::~Contraction↩ CandidateManager ( ) [inline]`

15.102.2 Member Function Documentation

15.102.2.1 candidates() `const std::vector<ContractionCandidate*>& smtrat::icp::Contraction↩ CandidateManager::candidates ( ) [inline]`

15.102.2.2 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

_candidate	

Returns the set of candidates in the closure of _candidate

15.102.2.3 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

_lhs	The slack/nonlinear variable which represents the constraint
_constraint	The constraint which is to be replaced
_derivationVar	The variable from which the derivative is created when performing contraction
_origin	The pointer to the original formula, needed for assertions and removals of subformulas

Returns a pointer to the created contraction candidate

15.102.2.4 getCandidate()

```
ContractionCandidate * smtrat::icp::ContractionCandidateManager↩
::getCandidate (
            unsigned _id ) const
```

Returns the contraction candidate for the given id.

Parameters

↩ ↩ *id*	

Returns the pointer to the contraction candidate

15.102.2.5 getId()

```
unsigned smtrat::icp::ContractionCandidateManager::getId (
            const ContractionCandidate *const _candidate ) const
```

Returns the id of the given contraction candidate.

Parameters

_candidate	

Returns id of the candidate

15.103 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

15.103.1 Member Function Documentation

15.103.1.1 operator()() [1/2] template<typename To>

```
template<typename From >
bool smtrat::parser::conversion::Converter< To >::operator() (
            const From & ,
            To &  ) const  [inline]
```

15.103.1.2 operator()() [2/2] template<typename To>

```
bool smtrat::parser::conversion::Converter< To >::operator() (
            const To & from,
            To & to ) const  [inline]
```

15.104 smtrat::parser::conversion::Converter< FormulaT > Struct Template 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

15.104.1 Member Function Documentation

15.104.1.1 operator()() [1/3] bool smtrat::parser::conversion::Converter< FormulaT >::operator() (

```
            const carl::Variable & from,
            FormulaT & to ) const  [inline]
```

15.104.1.2 operator()() [2/3] bool smtrat::parser::conversion::Converter< FormulaT >::operator() (

```
            const FormulaT & from,
            FormulaT & to ) const  [inline]
```

15.104.1.3 operator()() [3/3] template<typename From >

```
bool smtrat::parser::conversion::Converter< FormulaT >::operator() (
            const From & ,
            FormulaT &  ) const  [inline]
```

15.105 smtrat::parser::conversion::Converter< Poly > Struct Template 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

15.105.1 Member Function Documentation

15.105.1.1 operator()() [1/4]

```
bool smtrat::parser::conversion::Converter< Poly >::operator() (
    const carl::Variable & from,
    Poly & to ) const [inline]
```

15.105.1.2 operator()() [2/4]

```
template<typename From >
bool smtrat::parser::conversion::Converter< Poly >::operator() (
    const From & ,
    Poly & ) const [inline]
```

15.105.1.3 operator()() [3/4]

```
bool smtrat::parser::conversion::Converter< Poly >::operator() (
    const Poly & from,
    Poly & to ) const [inline]
```

15.105.1.4 operator()() [4/4]

```
bool smtrat::parser::conversion::Converter< Poly >::operator() (
    const Rational & from,
    Poly & to ) const [inline]
```

15.106 smtrat::parser::conversion::Converter< types::BVTerm > Struct Template 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

15.106.1 Member Function Documentation

15.106.1.1 operator()() [1/4]

```
bool smtrat::parser::conversion::Converter< types::BVTerm >↩
::operator() (
    const FixedWidthConstant< Integer > & from,
    types::BVTerm & to ) const [inline]
```

15.106.1.2 operator()() [2/4]

```
template<typename From >
bool smtrat::parser::conversion::Converter< types::BVTerm >::operator() (
    const From & ,
    types::BVTerm & ) const [inline]
```

15.106.1.3 operator()() [3/4]

```
bool smtrat::parser::conversion::Converter< types::BVTerm >↩
::operator() (
            const types::BVTerm & from,
            types::BVTerm & to ) const  [inline]
```

15.106.1.4 operator()() [4/4]

```
bool smtrat::parser::conversion::Converter< types::BVTerm >↩
::operator() (
            const types::BVVariable & from,
            types::BVTerm & to ) const  [inline]
```

15.107 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.

15.107.1 Detailed Description

Core settings.

15.107.2 Field Documentation

15.107.2.1 be_quiet

`bool benchmax::settings::CoreSettings::be_quiet`

Whether to be more quiet.

15.107.2.2 be_verbose

`bool benchmax::settings::CoreSettings::be_verbose`

Whether to be more verbose.

15.107.2.3 config_file

`std::string benchmax::settings::CoreSettings::config_file`

Path of an optional config file.

15.107.2.4 show_help `bool benchmax::settings::CoreSettings::show_help`

Whether to show the command-line help.

15.107.2.5 show_settings `bool benchmax::settings::CoreSettings::show_settings`

Whether to show the configured settings.

15.107.2.6 start_time `long benchmax::settings::CoreSettings::start_time = std::time(nullptr)`

Timestamp the binary was started.

15.108 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

15.108.1 Field Documentation

15.108.1.1 config_file `std::string smtrat::settings::CoreSettings::config_file`

15.108.1.2 show_cmake_options `bool smtrat::settings::CoreSettings::show_cmake_options`

15.108.1.3 show_help `bool smtrat::settings::CoreSettings::show_help`

15.108.1.4 show_info `bool smtrat::settings::CoreSettings::show_info`

15.108.1.5 show_license `bool smtrat::settings::CoreSettings::show_license`

15.108.1.6 show_settings `bool smtrat::settings::CoreSettings::show_settings`

15.108.1.7 show_strategy `bool smtrat::settings::CoreSettings::show_strategy`

15.108.1.8 show_version `bool smtrat::settings::CoreSettings::show_version`

15.109 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)
- 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 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

15.109.1 Detailed Description

Implements the core theory of the booleans.

15.109.2 Member Typedef Documentation

15.109.2.1 OperatorType `using smtrat::parser::CoreTheory::OperatorType = boost::variant<carl↩::FormulaType>`

15.109.3 Constructor & Destructor Documentation

15.109.3.1 CoreTheory() `smtrat::parser::CoreTheory::CoreTheory ( ParserState * state )`

15.109.4 Member Function Documentation

15.109.4.1 addConstants()

```
void smtrat::parser::CoreTheory::addConstants (
    qi::symbols< char, types::ConstType > & constants ) [static]
```

15.109.4.2 addSimpleSorts()

```
void smtrat::parser::CoreTheory::addSimpleSorts (
    qi::symbols< char, carl::Sort > & sorts ) [static]
```

15.109.4.3 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.

15.109.4.4 expandDistinct()

```
template<typename T , typename Builder >
FormulaT smtrat::parser::AbstractTheory::expandDistinct (
    const std::vector< T > & values,
    const Builder & neqBuilder ) [inline], [inherited]
```

15.109.4.5 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.

15.109.4.6 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.

15.109.4.7 handleITE()

```
bool smtrat::parser::CoreTheory::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.

15.109.4.8 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.

15.109.4.9 refreshVariable()

```
virtual bool smtrat::parser::AbstractTheory::refreshVariable (
    const types::VariableType & ,
    types::VariableType & ,
    TheoryError & errors )  [inline], [virtual], [inherited]
```

Reimplemented in smtrat::parser::BitvectorTheory.

15.109.4.10 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.

15.109.5 Field Documentation

15.109.5.1 state

```
ParserState* smtrat::parser::AbstractTheory::state  [inherited]
```

15.110 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

15.110.1 Detailed Description

Types of the core theory.

15.110.2 Member Typedef Documentation

15.110.2.1 ConstTypes

```
typedef mpl::vector<FormulaT, std::string> smtrat::parser::types::CoreTheory::ConstTyp
```

15.110.2.2 ExpressionTypes `typedef mpl::vector<FormulaT, std::string> smtrat::parser::types::CoreTheory::Exp`

15.110.2.3 TermType `typedef carl::mpl_variant_of<TermTypes>::type smtrat::parser::types::CoreTheory::TermType`

15.110.2.4 TermTypes `typedef mpl::vector<FormulaT, std::string> smtrat::parser::types::CoreTheory::TermTypes`

15.110.2.5 VariableTypes `typedef mpl::vector<carl::Variable> smtrat::parser::types::CoreTheory::VariableTypes`

15.111 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)

15.111.1 Member Function Documentation

15.111.1.1 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]
```

15.112 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)

15.112.1 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.

15.112.2 Constructor & Destructor Documentation

15.112.2.1 Covering()

```
smtrat::mcsat::arithmetic::Covering::Covering (
            std::size_t intervals )  [inline]
```

15.112.3 Member Function Documentation

15.112.3.1 add()

```
void smtrat::mcsat::arithmetic::Covering::add (
    const FormulaT & c,
    const carl::Bitset & b ) [inline]
```

15.112.3.2 buildConflictingCore()

```
void smtrat::mcsat::arithmetic::Covering::buildConflictingCore
(
    std::vector< FormulaT > & core ) const [inline]
```

15.112.3.3 conflicts()

```
bool smtrat::mcsat::arithmetic::Covering::conflicts ( ) const [inline]
```

15.112.3.4 satisfyingInterval()

```
std::size_t smtrat::mcsat::arithmetic::Covering::satisfying↩
Interval ( ) const [inline]
```

15.112.3.5 satisfyingSamples()

```
const auto& smtrat::mcsat::arithmetic::Covering::satisfying↩
Samples ( ) const [inline]
```

15.112.4 Friends And Related Function Documentation

15.112.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const Covering & ri ) [friend]
```

15.113 smtrat::cadcells::representation::covering< CoveringHeuristic::CHAIN_COVERING > Struct Template 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)

15.113.1 Member Function Documentation

15.113.1.1 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]
```

15.114 smtrat::cadcells::representation::covering< CoveringHeuristic::DEFAULT_COVERING > Struct Template 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)

15.114.1 Member Function Documentation

15.114.1.1 compute()

```
template<typename T >
static std::optional<datastructures::CoveringRepresentation<T> > smtrat::cadcells::representation::covering<
CoveringHeuristic::DEFAULT_COVERING >::compute (
            const std::vector< datastructures::SampledDerivationRef< T >> & ders ) [inline],
[static]
```

15.115 smtrat::mcsat::onecellcad::firstlevel::covering_at_level Struct Reference

```
#include <first_level.h>
```

Data Fields

- std::vector< cell_at_level > cells

15.115.1 Field Documentation

15.115.1.1 cells

```
std::vector<cell_at_level> smtrat::mcsat::onecellcad::firstlevel::covering_↩
at_level::cells
```

15.116 smtrat::cadcells::datastructures::CoveringDescription Class Reference

Describes a covering of the real line by CellDescriptions (given an appropriate sample).

```
#include <roots.h>
```

Public Member Functions

- void add (const CellDescription &c)

  Add a CellDescription to the covering.
- const auto & cells () const

15.116.1 Detailed Description

Describes a covering of the real line by CellDescriptions (given an appropriate sample).

15.116.2 Member Function Documentation

15.116.2.1 add()

```
void smtrat::cadcells::datastructures::CoveringDescription::add (
            const CellDescription & c ) [inline]
```

Add a CellDescription 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).

15.116.2.2 cells()

```
const auto& smtrat::cadcells::datastructures::CoveringDescription::cells ( )
const [inline]
```

15.117 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.

15.117.1 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.

15.117.2 Member Function Documentation

15.117.2.1 get_covering() `template<typename P>`

```
CoveringDescription smtrat::cadcells::datastructures::CoveringRepresentation< P >::get_↩
covering ( ) const  [inline]
```

Returns a descriptions of the covering.

15.117.2.2 is_valid() `template<typename P>`

```
bool smtrat::cadcells::datastructures::CoveringRepresentation< P >::is_valid ( ) const  [inline]
```

Checks whether this represents a proper non-redundant covering.

15.117.2.3 sampled_derivations() `template<typename P>`

```
std::vector<SampledDerivationRef<P> > smtrat::cadcells::datastructures::CoveringRepresentation<
P >::sampled_derivations ( )  [inline]
```

Returns the derivations.

15.117.3 Field Documentation

15.117.3.1 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.

15.117.3.2 ordering `template<typename P>`

`IndexedRootOrdering smtrat::cadcells::datastructures::CoveringRepresentation< P >::ordering`

An ordering on the roots for the cell boundaries mainting the covering.

15.118 smtrat::mcsat::onecellcad::recursive::CoverNullification Struct Reference

`#include <Explanation.h>`

Static Public Attributes

- static constexpr bool cover_nullification = true

15.118.1 Field Documentation

15.118.1.1 cover_nullification `constexpr bool smtrat::mcsat::onecellcad::recursive::CoverNullification::cover_nullification = true [static], [constexpr]`

15.119 smtrat::CSplitModule< Settings > Class Template Reference

`#include <CSplitModule.h>`

Public Types

- typedef Settings SettingsType
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::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 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 _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)

  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 _preferLeft↩ Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩ ::vector< FormulaT >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩ AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩ CaseWeak=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.

15.119.1 Member Typedef Documentation

15.119.1.1 SettingsType `template<typename Settings >`

```
typedef Settings smtrat::CSplitModule< Settings >::SettingsType
```

15.119.2 Member Enumeration Documentation

15.119.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

15.119.3 Constructor & Destructor Documentation

15.119.3.1 CSplitModule() `template<class Settings >`

```
smtrat::CSplitModule< Settings >::CSplitModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

15.119.4 Member Function Documentation

15.119.4.1 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.

15.119.4.2 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 >.

15.119.4.3 addCore()

```
template<class 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

_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.

Update the variable domain with the asserted bound

Normalize the left hand side of the constraint and turn the relation accordingly

Purifiy and discretize the normalized left hand side to construct the linearization

Check if the asserted relation trivially conflicts with other asserted relations

Reimplemented from smtrat::Module.

15.119.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.119.4.5 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.

15.119.4.6 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.

15.119.4.7 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.

15.119.4.8 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.

15.119.4.9 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.

15.119.4.10 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

_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.

15.119.4.11 allModels() `const std::vector<Model>& smtrat::Module::allModels ( ) const [inline], [inherited]`

Returns All satisfying assignments, if existent.

15.119.4.12 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.

15.119.4.13 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.

15.119.4.14 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.

15.119.4.15 branchAt() [1/6]

```
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]
```

15.119.4.16 branchAt() [2/6]

```
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]
```

15.119.4.17 branchAt() [3/6]

```
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]
```

15.119.4.18 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.119.4.19 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
    const typename P::PolyType & _poly,
    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]
```

15.119.4.20 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false )  [inline], [protected], [inherited]
```

15.119.4.21 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in smtrat::PModule.

15.119.4.22 checkCore()

```
template<class 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.

Report unsatisfiability if the already found conflicts are still unresolved

Apply the method only if the asserted formula is not trivially undecidable

Check the asserted formula with the backends

Reimplemented from smtrat::Module.

15.119.4.23 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
    std::vector< FormulaSetT >::const_iterator _infsubset,
    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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.119.4.24 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.

15.119.4.25 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.119.4.26 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.119.4.27 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.119.4.28 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.119.4.29 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.119.4.30 collectOrigins() [1/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

15.119.4.31 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

_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.

15.119.4.32 collectTheoryPropagations()

`void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.119.4.33 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ToInform ( ) const [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.119.4.34 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

15.119.4.35 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)

15.119.4.36 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

_constraint	The constraint to remove from internal data structures.

15.119.4.37 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.119.4.38 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.119.4.39 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.119.4.40 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.119.4.41 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.119.4.42 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.

15.119.4.43 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.

15.119.4.44 freeSplittingVariable()

```
static void smtrat::Module::freeSplittingVariable (
    const FormulaT & _splittingVariable ) [inline], [static], [inherited]
```

15.119.4.45 generateTrivialInfeasibleSubset()

```
void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]
```

Stores the trivial infeasible subset being the set of received formulas.

15.119.4.46 getBackendsAllModels()

```
void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]
```

Stores all models of a backend in the list of all models of this module.

15.119.4.47 getBackendsModel()

```
void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]
```

15.119.4.48 getInfeasibleSubsets() [1/2]

```
void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]
```

Copies the infeasible subsets of the passed formula.

15.119.4.49 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.119.4.50 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.119.4.51 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.

15.119.4.52 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.119.4.53 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.119.4.54 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.119.4.55 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 reveived formula, if the Boolean is true.

Reimplemented in smtrat::PModule.

15.119.4.56 hasLemmas()

`bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

15.119.4.57 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).

15.119.4.58 id()

`std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns A unique ID to identify this module instance.

15.119.4.59 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.

15.119.4.60 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.119.4.61 informBackends()

```
void smtrat::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.

15.119.4.62 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, and smtrat::ICPModule< ICPSettings1 >.

15.119.4.63 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.119.4.64 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::FPPModule< Settings >.

15.119.4.65 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.119.4.66 isLemmaLevel() `bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.119.4.67 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().

15.119.4.68 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.

15.119.4.69 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.119.4.70 model()

`const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.119.4.71 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

_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.

15.119.4.72 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.

15.119.4.73 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

15.119.4.74 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
    const FormulaT & _origin ) const [protected], [inherited]
```

15.119.4.75 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula. TODO: disable this method

15.119.4.76 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula.

15.119.4.77 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to the backends of this module.

15.119.4.78 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to this module.

15.119.4.79 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.119.4.80 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

_out	The stream to print the assignment on.

15.119.4.81 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const  [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.119.4.82 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

_out	The stream to print the assignment on.

15.119.4.83 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const  [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.119.4.84 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const  [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.119.4.85 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.

15.119.4.86 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

15.119.4.87 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩
InfeasibleSubset (
            ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

15.119.4.88 receivedVariable()

```
bool smtrat::Module::receivedVariable (
            carl::Variable::Arg _var ) const [inline], [inherited]
```

15.119.4.89 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

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

15.119.4.90 removeCore()

```
template<class Settings >
void smtrat::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

_subformula	The position of the subformula to remove.

Update the variable domain with the removed bound

Normalize the left hand side of the constraint and turn the relation accordingly

Retrieve the normalized constraint and mark the separator object as changed

Reimplemented from smtrat::Module.

15.119.4.91 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.119.4.92 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]
```

15.119.4.93 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.119.4.94 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.119.4.95 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]
```

Reimplemented in smtrat::PModule.

15.119.4.96 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

_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 in smtrat::PModule.

15.119.4.97 setId()

```
void smtrat::Module::setId ( std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.119.4.98 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.

15.119.4.99 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.

15.119.4.100 splitUnequalConstraint()

```
void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]
```

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.119.4.101 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.119.4.102 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 >.

15.119.4.103 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.119.4.104 updateModel() `template<class 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.

15.119.4.105 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).

15.119.5 Field Documentation

15.119.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.119.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.119.5.3 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.

15.119.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.119.5.5 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

15.119.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.119.5.7 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.

15.119.5.8 mFirstUncheckedReceivedSubformula `ModuleInput::const_iterator smtrat::Module::m↩FirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

15.119.5.9 mFoundAnswer `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.119.5.10 mFullCheck `bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.119.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.119.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.119.5.13 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.

15.119.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.119.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.119.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.119.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.119.5.18 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.

15.119.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplitting↩Variables [static], [inherited]`

Reusable splitting variables.

15.119.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.119.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.119.5.22 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.

15.119.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

15.119.5.24 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).

15.119.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.120 smtrat::CubeLIAModule< Settings > Class Template Reference

`#include <CubeLIAModule.h>`

Public Types

- typedef Settings SettingsType
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::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 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 _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)

  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 &_branching↩Value) 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 _preferLeft↩Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩::vector< FormulaT >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩CaseWeak=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.

15.120.1 Member Typedef Documentation

15.120.1.1 SettingsType `template<typename Settings >`

`typedef Settings smtrat::CubeLIAModule< Settings >::SettingsType`

15.120.2 Member Enumeration Documentation

15.120.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

15.120.3 Constructor & Destructor Documentation

15.120.3.1 CubeLIAModule()

```
template<class Settings >
smtrat::CubeLIAModule< Settings >::CubeLIAModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

15.120.3.2 ~CubeLIAModule()

```
template<class Settings >
smtrat::CubeLIAModule< Settings >::~CubeLIAModule ( )
```

15.120.4 Member Function Documentation

15.120.4.1 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.

15.120.4.2 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 >.

15.120.4.3 addCore()

```
template<class 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

_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.

15.120.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.120.4.5 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.

15.120.4.6 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.

15.120.4.7 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.

15.120.4.8 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.

15.120.4.9 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.

15.120.4.10 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

_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.

15.120.4.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns All satisfying assignments, if existent.

15.120.4.12 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.

15.120.4.13 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.

15.120.4.14 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.

15.120.4.15 branchAt() [1/6]

```
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]
```

15.120.4.16 branchAt() [2/6]

```
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]
```

15.120.4.17 branchAt() [3/6]

```
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]
```

15.120.4.18 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.120.4.19 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            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]
```

15.120.4.20 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

15.120.4.21 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in smtrat::PModule.

15.120.4.22 checkCore() `template<class Settings >`

`Answer smtrat::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 smtrat::Module.

15.120.4.23 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
    std::vector< FormulaSetT >::const_iterator _infsubset,
    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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.120.4.24 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.

15.120.4.25 cleanModel()

`void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.120.4.26 clearLemmas()

`void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.120.4.27 clearModel()

`void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.120.4.28 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.120.4.29 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.120.4.30 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

15.120.4.31 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

_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.

15.120.4.32 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.120.4.33 constraintsToInform() `const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.120.4.34 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

15.120.4.35 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)

15.120.4.36 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

_constraint	The constraint to remove from internal data structures.

15.120.4.37 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.120.4.38 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.120.4.39 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.120.4.40 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.120.4.41 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
            const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.120.4.42 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.

15.120.4.43 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.

15.120.4.44 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

15.120.4.45 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

15.120.4.46 getBackendsAllModels()

`void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

15.120.4.47 getBackendsModel()

`void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

15.120.4.48 getInfeasibleSubsets() [1/2]

`void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

15.120.4.49 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.120.4.50 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.120.4.51 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.

15.120.4.52 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.120.4.53 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.120.4.54 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.120.4.55 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 reveived formula, if the Boolean is true.

Reimplemented in smtrat::PModule.

15.120.4.56 hasLemmas() `bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

15.120.4.57 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).

15.120.4.58 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns A unique ID to identify this module instance.

15.120.4.59 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.

15.120.4.60 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.120.4.61 informBackends()

```
void smtrat::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.

15.120.4.62 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, and smtrat::ICPModule< ICPSettings1 >.

15.120.4.63 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.120.4.64 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::FPPModule< Settings >.

15.120.4.65 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.120.4.66 isLemmaLevel() `bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.120.4.67 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().

15.120.4.68 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.

15.120.4.69 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.120.4.70 model()

`const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.120.4.71 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

_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.

15.120.4.72 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.

15.120.4.73 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

15.120.4.74 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
    const FormulaT & _origin ) const [protected], [inherited]
```

15.120.4.75 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula. TODO: disable this method

15.120.4.76 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula.

15.120.4.77 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to the backends of this module.

15.120.4.78 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to this module.

15.120.4.79 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.120.4.80 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

_out	The stream to print the assignment on.

15.120.4.81 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const  [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.120.4.82 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

_out	The stream to print the assignment on.

15.120.4.83 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const  [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.120.4.84 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const  [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.120.4.85 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.

15.120.4.86 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

15.120.4.87 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩
InfeasibleSubset (
            ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

15.120.4.88 receivedVariable()

```
bool smtrat::Module::receivedVariable (
            carl::Variable::Arg _var ) const [inline], [inherited]
```

15.120.4.89 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

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

15.120.4.90 removeCore()

```
template<class Settings >
void smtrat::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

_subformula	The position of the subformula to remove.

Reimplemented from smtrat::Module.

15.120.4.91 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.120.4.92 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]
```

15.120.4.93 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.120.4.94 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.120.4.95 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]
```

Reimplemented in smtrat::PModule.

15.120.4.96 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

_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 in smtrat::PModule.

15.120.4.97 setId()

```
void smtrat::Module::setId (
    std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.120.4.98 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.

15.120.4.99 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.

15.120.4.100 splitUnequalConstraint()

```
void smtrat::Module::splitUnequalConstraint (
    const FormulaT & _unequalConstraint ) [protected], [inherited]
```

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.120.4.101 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.120.4.102 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 >.

15.120.4.103 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.120.4.104 updateModel() `template<class 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.

15.120.4.105 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).

15.120.5 Field Documentation

15.120.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.120.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.120.5.3 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.

15.120.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.120.5.5 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

15.120.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.120.5.7 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.

15.120.5.8 mFirstUncheckedReceivedSubformula `ModuleInput::const_iterator smtrat::Module::m↩FirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

15.120.5.9 mFoundAnswer `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.120.5.10 mFullCheck `bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.120.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.120.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.120.5.13 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.

15.120.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.120.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.120.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.120.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.120.5.18 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.

15.120.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplitting↩Variables [static], [inherited]`

Reusable splitting variables.

15.120.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.120.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.120.5.22 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.

15.120.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

15.120.5.24 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).

15.120.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.121 smtrat::CurryModule< Settings > Class Template Reference

```
#include <CurryModule.h>
```

Public Types

- typedef Settings SettingsType
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- 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 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_VARIAB↩LE)

  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 _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 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 &_branching↩Value) 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 _preferLeft↩Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩::vector< FormulaT >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩CaseWeak=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.

15.121.1 Member Typedef Documentation

15.121.1.1 SettingsType `template<typename Settings >`

`typedef Settings smtrat::CurryModule< Settings >::SettingsType`

15.121.2 Member Enumeration Documentation

15.121.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

15.121.3 Constructor & Destructor Documentation

15.121.3.1 CurryModule() template<class Settings >

```
smtrat::CurryModule< Settings >::CurryModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

15.121.3.2 ~CurryModule() template<class Settings >

```
smtrat::CurryModule< Settings >::~CurryModule ( )
```

15.121.4 Member Function Documentation

15.121.4.1 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.

15.121.4.2 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 >.

15.121.4.3 addCore() template<class 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

_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.

15.121.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.121.4.5 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.

15.121.4.6 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.

15.121.4.7 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.

15.121.4.8 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.

15.121.4.9 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.

15.121.4.10 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

_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.

15.121.4.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns All satisfying assignments, if existent.

15.121.4.12 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.

15.121.4.13 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.

15.121.4.14 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.

15.121.4.15 branchAt() [1/6]

```
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]
```

15.121.4.16 branchAt() [2/6]

```
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]
```

15.121.4.17 branchAt() [3/6]

```
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]
```

15.121.4.18 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.121.4.19 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            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]
```

15.121.4.20 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

15.121.4.21 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in smtrat::PModule.

15.121.4.22 checkCore()

```
template<class Settings >
Answer smtrat::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 smtrat::Module.

15.121.4.23 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
    std::vector< FormulaSetT >::const_iterator _infsubset,
    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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.121.4.24 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.

15.121.4.25 cleanModel()

```
void smtrat::Module::cleanModel ( ) const [inline], [protected],
[inherited]
```

Substitutes variable occurrences by its model value in the model values of other variables.

15.121.4.26 clearLemmas()

```
void smtrat::Module::clearLemmas ( ) [inline], [inherited]
```

Deletes all yet found lemmas.

15.121.4.27 clearModel()

```
void smtrat::Module::clearModel ( ) const [inline], [protected],
[inherited]
```

Clears the assignment, if any was found.

15.121.4.28 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.121.4.29 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.121.4.30 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

15.121.4.31 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

_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.

15.121.4.32 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.121.4.33 constraintsToInform() `const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.121.4.34 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

15.121.4.35 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)

15.121.4.36 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

_constraint	The constraint to remove from internal data structures.

15.121.4.37 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.121.4.38 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.121.4.39 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.121.4.40 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.121.4.41 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
            const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.121.4.42 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.

15.121.4.43 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.

15.121.4.44 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

15.121.4.45 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

15.121.4.46 getBackendsAllModels()

`void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

15.121.4.47 getBackendsModel()

`void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

15.121.4.48 getInfeasibleSubsets() [1/2]

`void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

15.121.4.49 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.121.4.50 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.121.4.51 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.

15.121.4.52 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.121.4.53 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.121.4.54 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.121.4.55 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 reveived formula, if the Boolean is true.

Reimplemented in smtrat::PModule.

15.121.4.56 hasLemmas() `bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

15.121.4.57 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).

15.121.4.58 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns A unique ID to identify this module instance.

15.121.4.59 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.

15.121.4.60 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.121.4.61 informBackends()

```
void smtrat::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.

15.121.4.62 informCore()

```
template<class 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

_constraint	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.

15.121.4.63 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.121.4.64 init()

```
template<class 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.

15.121.4.65 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.121.4.66 isLemmaLevel() `bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.121.4.67 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().

15.121.4.68 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.

15.121.4.69 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.121.4.70 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.121.4.71 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

_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.

15.121.4.72 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.

15.121.4.73 objective()

```
carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]
```

15.121.4.74 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
    const FormulaT & _origin ) const [protected], [inherited]
```

15.121.4.75 passedFormulaBegin()

```
ModuleInput::iterator smtrat::Module::passedFormulaBegin ( )
[inline], [protected], [inherited]
```

Returns An iterator to the end of the passed formula. TODO: disable this method

15.121.4.76 passedFormulaEnd()

```
ModuleInput::iterator smtrat::Module::passedFormulaEnd ( )
[inline], [protected], [inherited]
```

Returns An iterator to the end of the passed formula.

15.121.4.77 pPassedFormula()

```
const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline],
[inherited]
```

Returns A pointer to the formula passed to the backends of this module.

15.121.4.78 pReceivedFormula()

```
const ModuleInput* smtrat::Module::pReceivedFormula ( ) const
[inline], [inherited]
```

Returns A pointer to the formula passed to this module.

15.121.4.79 print()

```
void smtrat::Module::print (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.121.4.80 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

_out	The stream to print the assignment on.

15.121.4.81 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.121.4.82 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

_out	The stream to print the assignment on.

15.121.4.83 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.121.4.84 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.121.4.85 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.

15.121.4.86 receivedFormulaChecked()

```
void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]
```

Notifies that the received formulas has been checked.

15.121.4.87 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩InfeasibleSubset (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

15.121.4.88 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const [inline], [inherited]
```

15.121.4.89 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

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

15.121.4.90 removeCore()

```
template<class Settings >
void smtrat::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

_subformula	The position of the subformula to remove.

Reimplemented from smtrat::Module.

15.121.4.91 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin
(
            ModuleInput::iterator _formula,
            const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.121.4.92 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

15.121.4.93 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline],
[inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.121.4.94 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.121.4.95 runBackends() [1/2]

`virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]`

Reimplemented in smtrat::PModule.

15.121.4.96 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

_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 in smtrat::PModule.

15.121.4.97 setId()

```
void smtrat::Module::setId (
    std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.121.4.98 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.

15.121.4.99 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.

15.121.4.100 splitUnequalConstraint()

`void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.121.4.101 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.121.4.102 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 >.

15.121.4.103 updateLemmas()

`void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.121.4.104 updateModel()

`template<class 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.

15.121.4.105 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).

15.121.5 Field Documentation

15.121.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.121.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.121.5.3 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.

15.121.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.121.5.5 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

15.121.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.121.5.7 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.

15.121.5.8 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.

15.121.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.121.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.121.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.121.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.121.5.13 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.

15.121.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.121.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.121.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.121.5.17 mNumOfBranchVarsToStore `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.121.5.18 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.

15.121.5.19 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

15.121.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.121.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.121.5.22 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.

15.121.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

15.121.5.24 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).

15.121.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.122 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).

15.122.1 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.

15.122.2 Constructor & Destructor Documentation

15.122.2.1 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.

15.122.3 Member Function Documentation

15.122.3.1 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).

15.123 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.

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

15.123.2 Member Typedef Documentation

15.123.2.1 Index

```
using benchmax::Database::Index = std::size_t
```

Dummy index type.

15.123.3 Member Function Documentation

15.123.3.1 addBenchmarkAttribute()

```
void benchmax::Database::addBenchmarkAttribute (
    Index ,
    const std::string & ,
    const std::string & ) [inline]
```

Dummy.

15.123.3.2 addBenchmarkResult()

```
Index benchmax::Database::addBenchmarkResult (
    Index ,
    Index ,
    Index ,
    int ,
    std::size_t ) [inline]
```

Dummy.

15.123.3.3 addFile()

```
Index benchmax::Database::addFile (
    const fs::path & ) [inline]
```

Dummy.

15.123.3.4 addTool()

```
Index benchmax::Database::addTool (
    const Tool * ) [inline]
```

Dummy.

15.123.3.5 createBenchmark()

```
Index benchmax::Database::createBenchmark ( ) [inline]
```

Dummy.

15.123.3.6 getFileID()

```
Index benchmax::Database::getFileID (
    const fs::path & ) [inline]
```

Dummy.

15.123.3.7 getToolID()

```
Index benchmax::Database::getToolID (
    const Tool * ) [inline]
```

Dummy.

15.124 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)

15.124.1 Member Typedef Documentation

15.124.1.1 Index [1/2] `typedef mysqlpp::sql_bigint_unsigned benchmax::DBAL::Index`

15.124.1.2 Index [2/2] `using benchmax::DBAL::Index = std::size_t`

Indec type.

15.124.1.3 Results [1/2] `typedef mysqlpp::StoreQueryResult benchmax::DBAL::Results`

15.124.1.4 Results [2/2] `using benchmax::DBAL::Results = std::unique_ptr<sql::ResultSet>`

Results type.

15.124.1.5 Statement [1/2] `typedef mysqlpp::Query benchmax::DBAL::Statement`

15.124.1.6 Statement [2/2] `using benchmax::DBAL::Statement = std::unique_ptr<sql::Prepared↩ Statement>`

Statement type.

15.124.2 Member Function Documentation

15.124.2.1 connect() [1/2]

```
bool benchmax::DBAL::connect (
    const std::string & db,
    const std::string & host,
    const std::string & user,
    const std::string & password ) [inline]
```

15.124.2.2 connect() [2/2]

```
bool benchmax::DBAL::connect (
    const std::string & db,
    const std::string & host,
    const std::string & user,
    const std::string & password ) [inline]
```

Connect to the given database.

15.124.2.3 execute() [1/6]

```
Results benchmax::DBAL::execute (
    Statement & stmt ) [inline]
```

Execute a statement without arguments.

15.124.2.4 execute() [2/6]

```
template<typename... Args>
void benchmax::DBAL::execute (
    Statement & stmt,
    const Args &... args ) [inline]
```

15.124.2.5 execute() [3/6]

```
Results benchmax::DBAL::execute (
    Statement & stmt,
    const std::string & a1 ) [inline]
```

Execute a statement with a single string.

15.124.2.6 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.

15.124.2.7 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.

15.124.2.8 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.

15.124.2.9 getIndex() [1/2]

```
Index benchmax::DBAL::getIndex (
    const Results & res,
    const std::string & name ) [inline]
```

15.124.2.10 getIndex() [2/2]

```
Index benchmax::DBAL::getIndex (
    const Results & res,
    const std::string & name ) [inline]
```

Get the index of some name within the results.

15.124.2.11 insert() [1/2]

```
template<typename... Args>
Index benchmax::DBAL::insert (
    const std::string & query,
    const Args &... args ) [inline]
```

15.124.2.12 insert() [2/2]

```
template<typename... Args>
Index benchmax::DBAL::insert (
    const std::string & query,
    const Args &... args ) [inline]
```

Insert some data and return the new index.

15.124.2.13 prepare() [1/2]

```
Statement benchmax::DBAL::prepare (
    const std::string & query ) [inline]
```

15.124.2.14 prepare() [2/2]

```
Statement benchmax::DBAL::prepare (
    const std::string & query ) [inline]
```

Prepare a statement.

15.124.2.15 select() [1/2]

```
template<typename... Args>
Results benchmax::DBAL::select (
    const std::string & query,
    const Args &... args ) [inline]
```

15.124.2.16 select() [2/2] template<typename... Args>

```
Results benchmax::DBAL::select (
            const std::string & query,
            const Args &... args ) [inline]
```

Select some data.

15.124.2.17 size() [1/2] std::size_t benchmax::DBAL::size (

```
            const Results & res ) [inline]
```

15.124.2.18 size() [2/2] std::size_t benchmax::DBAL::size (

```
            const Results & res ) [inline]
```

Return the size of the results.

15.125 smtrat::parser::DecimalParser Struct Reference

Parses decimals: `numeral.0*numeral`

```
#include <Lexicon.h>
```

15.125.1 Detailed Description

Parses decimals: `numeral.0*numeral`

15.126 Minisat::DeepEqual< K > Struct Template Reference

```
#include <Map.h>
```

Public Member Functions

- bool operator() (const K *k1, const K *k2) const

15.126.1 Member Function Documentation

15.126.1.1 operator()() template<class K >

```
bool Minisat::DeepEqual< K >::operator() (
            const K * k1,
            const K * k2 ) const [inline]
```

15.127 Minisat::DeepHash< K > Struct Template Reference

```
#include <Map.h>
```

Public Member Functions

- uint32_t operator() (const K *k) const

15.127.1 Member Function Documentation

15.127.1.1 operator()() template<class K >

```
uint32_t Minisat::DeepHash< K >::operator() (
            const K * k ) const [inline]
```

15.128 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

15.128.1 Detailed Description

Does not order anything.

15.128.2 Member Function Documentation

15.128.2.1 operator()()

```
bool smtrat::mcsat::fm::DefaultComparator::operator() (
    const Bound & ,
    const Bound & ) const [inline]
```

15.128.3 Field Documentation

15.128.3.1 symmetric

```
bool smtrat::mcsat::fm::DefaultComparator::symmetric = false
```

15.129 smtrat::mcsat::vs::DefaultSettings Struct Reference

```
#include <ExplanationGenerator.h>
```

Static Public Attributes

- static const bool reduceConflictConstraints = true
- static const bool clauseChainWithEquivalences = false

15.129.1 Field Documentation

15.129.1.1 clauseChainWithEquivalences

```
const bool smtrat::mcsat::vs::DefaultSettings::clause↩
ChainWithEquivalences = false [static]
```

15.129.1.2 reduceConflictConstraints

```
const bool smtrat::mcsat::vs::DefaultSettings::reduce↩
ConflictConstraints = true [static]
```

15.130 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.

15.130.1 Field Documentation

15.130.1.1 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.

15.130.1.2 assignAllVariables

`constexpr unsigned int smtrat::mcsat::smtaf::DefaultSettings↩ ::assignAllVariables = true [static], [constexpr]`

15.131 smtrat::mcsat::fm::DefaultSettings Struct Reference

`#include <Explanation.h>`

Static Public Attributes

- static constexpr bool use_all_constraints = false

15.131.1 Field Documentation

15.131.1.1 use_all_constraints

`constexpr bool smtrat::mcsat::fm::DefaultSettings::use_all_↩ constraints = false [static], [constexpr]`

15.132 smtrat::cad::projection_compare::degree Struct Reference

`#include <ProjectionComparator.h>`

15.133 smtrat::mcsat::onecellcad::recursive::DegreeAscending Struct Reference

`#include <Explanation.h>`

Static Public Attributes

- static constexpr int heuristic = 1

15.133.1 Field Documentation

15.133.1.1 heuristic

`constexpr int smtrat::mcsat::onecellcad::recursive::DegreeAscending↩ ::heuristic = 1 [static], [constexpr]`

15.134 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

15.134.1 Member Function Documentation

15.134.1.1 operator()() [1/3]

```
void smtrat::analyzer::DegreeCollector::operator() (
    const carl::UnivariatePolynomial< Poly > & p ) [inline]
```

15.134.1.2 operator()() [2/3]

```
void smtrat::analyzer::DegreeCollector::operator() (
    const ConstraintT & c ) [inline]
```

15.134.1.3 operator()() [3/3] `void smtrat::analyzer::DegreeCollector::operator() ( const Poly & p ) [inline]`

15.134.2 Field Documentation

15.134.2.1 degree_max `std::size_t smtrat::analyzer::DegreeCollector::degree_max = 0`

15.134.2.2 degree_sum `std::size_t smtrat::analyzer::DegreeCollector::degree_sum = 0`

15.134.2.3 tdegree_max `std::size_t smtrat::analyzer::DegreeCollector::tdegree_max = 0`

15.134.2.4 tdegree_sum `std::size_t smtrat::analyzer::DegreeCollector::tdegree_sum = 0`

15.135 smtrat::mcsat::onecellcad::recursive::DegreeDescending Struct Reference

`#include <Explanation.h>`

Static Public Attributes
- static constexpr int heuristic = 2

15.135.1 Field Documentation

15.135.1.1 heuristic `constexpr int smtrat::mcsat::onecellcad::recursive::DegreeDescending↩ ::heuristic = 2 [static], [constexpr]`

15.136 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)
- BaseDerivationRef< Properties > & base ()
- BaseDerivationRef< Properties > & base () const
- Delineation & delin ()
- const Assignment & underlying_sample () const
- DerivationRef< Properties > & underlying ()
- 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)

15.136.1 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

Properties	Set of properties

15.136.2 Constructor & Destructor Documentation

15.136.2.1 DelineatedDerivation()

```
template<typename Properties>
smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::DelineatedDerivation (
            BaseDerivationRef< Properties > base ) [inline]
```

15.136.3 Member Function Documentation

15.136.3.1 base() [1/2]

```
template<typename Properties>
BaseDerivationRef<Properties>& smtrat::cadcells::datastructures::DelineatedDerivation< Properties
>::base ( ) [inline]
```

15.136.3.2 base() [2/2]

```
template<typename Properties>
BaseDerivationRef<Properties>& smtrat::cadcells::datastructures::DelineatedDerivation< Properties
>::base ( ) const [inline]
```

15.136.3.3 contains()

```
template<typename Properties>
template<typename P >
bool smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::contains (
            const P & property ) const [inline]
```

15.136.3.4 delin()

```
template<typename Properties>
Delineation& smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::delin ( )
[inline]
```

15.136.3.5 insert()

```
template<typename Properties>
template<typename P >
void smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::insert (
            P property ) [inline]
```

15.136.3.6 level()

```
template<typename Properties>
size_t smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::level ( ) const
[inline]
```

15.136.3.7 main_var()

```
template<typename Properties>
carl::Variable smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::main_var
( ) const [inline]
```

15.136.3.8 merge_with() template<typename Properties>

```
void smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::merge_with (
            const DelineatedDerivation< Properties > & other ) [inline]
```

15.136.3.9 polys() template<typename Properties>

```
PolyPool& smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::polys ( )
[inline]
```

15.136.3.10 proj() template<typename Properties>

```
Projections& smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::proj ( )
[inline]
```

15.136.3.11 properties() template<typename Properties>

```
template<typename P >
const PropertiesTSet<P>& smtrat::cadcells::datastructures::DelineatedDerivation< Properties
>::properties ( ) const [inline]
```

15.136.3.12 underlying() [1/2] template<typename Properties>

```
DerivationRef<Properties>& smtrat::cadcells::datastructures::DelineatedDerivation< Properties
>::underlying ( ) [inline]
```

15.136.3.13 underlying() [2/2] template<typename Properties>

```
DerivationRef<Properties>& smtrat::cadcells::datastructures::DelineatedDerivation< Properties
>::underlying ( ) const [inline]
```

15.136.3.14 underlying_sample() template<typename Properties>

```
const Assignment& smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::underlying↩
_sample ( ) const [inline]
```

15.137 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, IndexedRoot ir_root)
- void add_poly_noroot (PolyRef poly)
- void add_poly_nullified (PolyRef poly)

Friends

- class DelineationInterval

15.137.1 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.

15.137.2 Constructor & Destructor Documentation

15.137.2.1 Delineation()

```
smtrat::cadcells::datastructures::Delineation::Delineation ( ) [inline]
```

15.137.3 Member Function Documentation

15.137.3.1 add_poly_noroot()

```
void smtrat::cadcells::datastructures::Delineation::add_poly_noroot (
    PolyRef poly ) [inline]
```

15.137.3.2 add_poly_nullified()

```
void smtrat::cadcells::datastructures::Delineation::add_poly_↩
nullified (
    PolyRef poly ) [inline]
```

15.137.3.3 add_root()

```
void smtrat::cadcells::datastructures::Delineation::add_root (
    RAN root,
    IndexedRoot ir_root ) [inline]
```

15.137.3.4 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.

15.137.3.5 empty()

```
bool smtrat::cadcells::datastructures::Delineation::empty ( ) const [inline]
```

15.137.3.6 nonzero()

```
const auto& smtrat::cadcells::datastructures::Delineation::nonzero ( )
const [inline]
```

The set of polynomials without roots.

15.137.3.7 nullified()

```
const auto& smtrat::cadcells::datastructures::Delineation::nullified ( )
const [inline]
```

The set of nullified polynomials.

15.137.3.8 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.

15.137.4 Friends And Related Function Documentation

15.137.4.1 DelineationInterval `friend class DelineationInterval [friend]`

15.138 smtrat::cadcells::representation::delineation< H > Struct Template Reference

`#include <heuristics.h>`

Static Public Member Functions

- template<typename T >
  static std::optional< datastructures::DelineationRepresentation< T > > compute (datastructures::DelineatedDerivationRef< T > &der)

15.138.1 Member Function Documentation

15.138.1.1 compute()

```
template<DelineationHeuristic H>
template<typename T >
static std::optional<datastructures::DelineationRepresentation<T> > smtrat::cadcells::representation::deline
H >::compute (
            datastructures::DelineatedDerivationRef< T > & der )  [static]
```

15.139 smtrat::cadcells::representation::delineation< DelineationHeuristic::CHAIN > Struct Template Reference

`#include <heuristics_delineation.h>`

Static Public Member Functions

- template<typename T >
  static std::optional< datastructures::DelineationRepresentation< T > > compute (datastructures::DelineatedDerivationRef< T > &der)

15.139.1 Member Function Documentation

15.139.1.1 compute()

```
template<typename T >
static std::optional<datastructures::DelineationRepresentation<T> > smtrat::cadcells::representation::deline
DelineationHeuristic::CHAIN >::compute (
            datastructures::DelineatedDerivationRef< T > & der )  [inline], [static]
```

15.140 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
- const auto & upper () const
- bool upper_unbounded () const

Friends

- class Delineation

15.140.1 Detailed Description An interval of a delineation.

15.140.2 Member Function Documentation

15.140.2.1 is_section() `bool smtrat::cadcells::datastructures::DelineationInterval::is_section ( ) const [inline]`

15.140.2.2 is_sector() `bool smtrat::cadcells::datastructures::DelineationInterval::is_sector ( ) const [inline]`

15.140.2.3 lower() `const auto& smtrat::cadcells::datastructures::DelineationInterval::lower ( ) const [inline]`

15.140.2.4 lower_unbounded() `bool smtrat::cadcells::datastructures::DelineationInterval↩ ::lower_unbounded ( ) const [inline]`

15.140.2.5 upper() `const auto& smtrat::cadcells::datastructures::DelineationInterval::upper ( ) const [inline]`

15.140.2.6 upper_unbounded() `bool smtrat::cadcells::datastructures::DelineationInterval↩ ::upper_unbounded ( ) const [inline]`

15.140.3 Friends And Related Function Documentation

15.140.3.1 Delineation `friend class Delineation [friend]`

15.141 smtrat::cadcells::datastructures::DelineationRepresentation< P > Struct Template Reference

Represents a delineation.

```
#include <representation.h>
```

Public Member Functions

- DelineationRepresentation (DelineatedDerivationRef< P > deriv)

Data Fields

- IndexedRootOrdering ordering

  An ordering of the roots that represents the delineation.
- DelineatedDerivationRef< P > derivation

  Derivation.

15.141.1 Detailed Description

template<typename P>
struct smtrat::cadcells::datastructures::DelineationRepresentation< P >

Represents a delineation.

15.141.2 Constructor & Destructor Documentation

15.141.2.1 DelineationRepresentation()

```
template<typename P>
smtrat::cadcells::datastructures::DelineationRepresentation< P >::DelineationRepresentation (
            DelineatedDerivationRef< P > deriv ) [inline]
```

15.141.3 Field Documentation

15.141.3.1 derivation

```
template<typename P>
DelineatedDerivationRef<P> smtrat::cadcells::datastructures::DelineationRepresentation< P >↩
::derivation
```

Derivation.

15.141.3.2 ordering

```
template<typename P>
IndexedRootOrdering smtrat::cadcells::datastructures::DelineationRepresentation< P >::ordering
```

An ordering of the roots that represents the delineation.

15.142 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

15.142.1 Member Function Documentation

15.142.1.1 add() [1/2]

```
void smtrat::mcsat::icp::Dependencies::add (
            carl::Variable v,
            const FormulaT & c,
            const std::vector< carl::Variable > & used ) [inline]
```

15.142.1.2 add() [2/2]

```
template<typename Contractor >
void smtrat::mcsat::icp::Dependencies::add (
            const Contractor & c ) [inline]
```

15.142.1.3 get()

```
std::vector<FormulaT> smtrat::mcsat::icp::Dependencies::get (
            carl::Variable v,
            bool negate = true ) const [inline]
```

15.143 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)

15.143.1 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)

15.143.1.1 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.

15.143.2 Constructor & Destructor Documentation

15.143.2.1 DerivationRef() [1/3] template<typename Properties>

```
smtrat::cadcells::datastructures::DerivationRef< Properties >::DerivationRef (
            const BaseDerivationRef< Properties > & data ) [inline]
```

15.143.2.2 DerivationRef() [2/3] template<typename Properties>

```
smtrat::cadcells::datastructures::DerivationRef< Properties >::DerivationRef (
            const DelineatedDerivationRef< Properties > & data ) [inline]
```

15.143.2.3 DerivationRef() [3/3] template<typename Properties>

```
smtrat::cadcells::datastructures::DerivationRef< Properties >::DerivationRef (
            const SampledDerivationRef< Properties > & data ) [inline]
```

15.143.3 Member Function Documentation

15.143.3.1 base() [1/2] template<typename Properties>

```
auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::base ( ) [inline]
```

15.143.3.2 base() [2/2] template<typename Properties>

```
const auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::base ( ) const
[inline]
```

15.143.3.3 base_ref() [1/2] template<typename Properties>

```
auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::base_ref ( ) [inline]
```

15.143.3.4 base_ref() [2/2] template<typename Properties>

```
const auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::base_ref ( ) const
[inline]
```

15.143.3.5 delineated() [1/2] template<typename Properties>

```
auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::delineated ( ) [inline]
```

15.143.3.6 delineated() [2/2] template<typename Properties>

```
const auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::delineated ( )
const [inline]
```

15.143.3.7 delineated_ref() [1/2] template<typename Properties>

```
auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::delineated_ref ( ) [inline]
```

15.143.3.8 delineated_ref() [2/2] template<typename Properties>

```
const auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::delineated_ref ( )
const [inline]
```

15.143.3.9 is_null()

```
template<typename Properties>
bool smtrat::cadcells::datastructures::DerivationRef< Properties >::is_null ( ) const [inline]
```

15.143.3.10 is_sampled()

```
template<typename Properties>
bool smtrat::cadcells::datastructures::DerivationRef< Properties >::is_sampled ( ) const [inline]
```

15.143.3.11 sampled() [1/2]

```
template<typename Properties>
auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::sampled ( ) [inline]
```

15.143.3.12 sampled() [2/2]

```
template<typename Properties>
const auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::sampled ( ) const
[inline]
```

15.143.3.13 sampled_ref() [1/2]

```
template<typename Properties>
auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::sampled_ref ( ) [inline]
```

15.143.3.14 sampled_ref() [2/2]

```
template<typename Properties>
const auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::sampled_ref ( )
const [inline]
```

15.143.4 Friends And Related Function Documentation

15.143.4.1 operator<

```
template<typename Properties>
template<typename P >
bool operator< (
            const DerivationRef< P > & lhs,
            const DerivationRef< P > & rhs ) [friend]
```

15.143.4.2 operator==

```
template<typename Properties>
template<typename P >
bool operator== (
            const DerivationRef< P > & lhs,
            const DerivationRef< P > & rhs ) [friend]
```

15.144 smtrat::mcsat::onecellcad::recursive::DontCoverNullification Struct Reference

```
#include <Explanation.h>
```

Static Public Attributes

- static constexpr bool cover_nullification = false

15.144.1 Field Documentation

15.144.1.1 cover_nullification

```
constexpr bool smtrat::mcsat::onecellcad::recursive::DontCover↩
Nullification::cover_nullification = false [static], [constexpr]
```

15.145 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)

15.145.1 Constructor & Destructor Documentation

15.145.1.1 DotSubgraph()

```
smtrat::cad::debug::DotSubgraph::DotSubgraph (
    const std::string & n ) [inline]
```

15.145.2 Member Function Documentation

15.145.2.1 add()

```
void smtrat::cad::debug::DotSubgraph::add (
    const std::string & n ) [inline]
```

15.145.3 Friends And Related Function Documentation

15.145.3.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const DotSubgraph & dsg ) [friend]
```

15.145.4 Field Documentation

15.145.4.1 name

```
std::string smtrat::cad::debug::DotSubgraph::name
```

15.145.4.2 nodes

```
std::vector<std::string> smtrat::cad::debug::DotSubgraph::nodes
```

15.146 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

15.146.1 Constructor & Destructor Documentation

15.146.1.1 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]
```

15.146.2 Member Function Documentation

15.146.2.1 getHelpPrefixString()

```
static const char*& Minisat::Option::getHelpPrefixString ( ) [inline], [static], [protected], [inherited]
```

15.146.2.2 getOptionList()

```
static vec<Option*>& Minisat::Option::getOptionList ( ) [inline], [static], [protected], [inherited]
```

15.146.2.3 getUsageString()

```
static const char*& Minisat::Option::getUsageString ( ) [inline], [static], [protected], [inherited]
```

15.146.2.4 help()

```
virtual void Minisat::DoubleOption::help (
    bool verbose = false ) [inline], [virtual]
```

Implements Minisat::Option.

15.146.2.5 operator double()

```
Minisat::DoubleOption::operator double (
    void ) const [inline]
```

15.146.2.6 operator=()

```
DoubleOption& Minisat::DoubleOption::operator= (
    double x ) [inline]
```

15.146.3 Field Documentation

15.146.3.1 category

```
const char* Minisat::Option::category [protected], [inherited]
```

15.146.3.2 description

```
const char* Minisat::Option::description [protected], [inherited]
```

15.146.3.3 name `const char* Minisat::Option::name [protected], [inherited]`

15.146.3.4 range `DoubleRange Minisat::DoubleOption::range [protected]`

15.146.3.5 type_name `const char* Minisat::Option::type_name [protected], [inherited]`

15.146.3.6 value `double Minisat::DoubleOption::value [protected]`

15.147 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

15.147.1 Constructor & Destructor Documentation

15.147.1.1 DoubleRange()

```
Minisat::DoubleRange::DoubleRange (
    double b,
    bool binc,
    double e,
    bool einc ) [inline]
```

15.147.2 Field Documentation

15.147.2.1 begin `double Minisat::DoubleRange::begin`

15.147.2.2 begin_inclusive `bool Minisat::DoubleRange::begin_inclusive`

15.147.2.3 end `double Minisat::DoubleRange::end`

15.147.2.4 end_inclusive `bool Minisat::DoubleRange::end_inclusive`

15.148 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

15.148.1 Member Function Documentation

15.148.1.1 operator()() [1/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const BinaryExpression & expr ) const  [inline], [inherited]
```

15.148.1.2 operator()() [2/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const carl::Variable & expr ) const  [inline], [inherited]
```

15.148.1.3 operator()() [3/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const ExpressionContent * _ec ) const  [inline], [inherited]
```

15.148.1.4 operator()() [4/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const ITEExpression & expr ) const  [inline], [inherited]
```

15.148.1.5 operator()() [5/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const NaryExpression & expr ) const  [inline], [inherited]
```

15.148.1.6 operator()() [6/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const QuantifierExpression & expr ) const  [inline], [inherited]
```

15.148.1.7 operator()() [7/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const UnaryExpression & expr ) const  [inline], [inherited]
```

15.148.1.8 simplify() [1/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::↩
BaseSimplifier::simplify (
            const BinaryExpression &  ) const  [inline], [protected], [virtual], [inherited]
```

15.148.1.9 simplify() [2/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::↩
BaseSimplifier::simplify (
            const carl::Variable &  ) const  [inline], [protected], [virtual], [inherited]
```

15.148.1.10 simplify() [3/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::↩
BaseSimplifier::simplify (
            const ITEExpression &  ) const  [inline], [protected], [virtual], [inherited]
```

15.148.1.11 simplify() [4/6]

```
const ExpressionContent* smtrat::expression::simplifier::Duplicate↩
Simplifier::simplify (
            const NaryExpression & expr ) const  [inline], [virtual]
```

Reimplemented from smtrat::expression::simplifier::BaseSimplifier.

15.148.1.12 simplify() [5/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::↩
BaseSimplifier::simplify (
            const QuantifierExpression &  ) const  [inline], [protected], [virtual], [inherited]
```

15.148.1.13 simplify() [6/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::↩
BaseSimplifier::simplify (
            const UnaryExpression &  ) const  [inline], [protected], [virtual], [inherited]
```

Reimplemented in smtrat::expression::simplifier::NegationSimplifier.

15.149 smtrat::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

15.149.1 Member Typedef Documentation

15.149.1.1 value_type

```
template<typename T , typename Compare = std::less<T>>
using smtrat::DynamicPriorityQueue< T, Compare >::value_type = typename Sequence::value_type
```

15.149.2 Constructor & Destructor Documentation

15.149.2.1 DynamicPriorityQueue() [1/2]

```
template<typename T , typename Compare = std::less<↩
T>>
smtrat::DynamicPriorityQueue< T, Compare >::DynamicPriorityQueue (
            const Compare & _comp = Compare() )  [inline], [explicit]
```

15.149.2.2 DynamicPriorityQueue() [2/2]

```
template<typename T , typename Compare = std::less<↩
T>>
smtrat::DynamicPriorityQueue< T, Compare >::DynamicPriorityQueue (
            Sequence && _seq,
            const Compare & _comp = Compare() )  [inline], [explicit]
```

15.149.3 Member Function Documentation

15.149.3.1 emplace()

```
template<typename T , typename Compare = std::less<T>>
template<typename...  Args>
void smtrat::DynamicPriorityQueue< T, Compare >::emplace (
            Args &&...  args )  [inline]
```

15.149.3.2 empty()

```
template<typename T , typename Compare = std::less<T>>
bool smtrat::DynamicPriorityQueue< T, Compare >::empty ( ) const  [inline]
```

15.149.3.3 fix()

```
template<typename T , typename Compare = std::less<T>>
void smtrat::DynamicPriorityQueue< T, Compare >::fix ( )  [inline]
```

15.149.3.4 pop()

```
template<typename T , typename Compare = std::less<T>>
void smtrat::DynamicPriorityQueue< T, Compare >::pop ( )  [inline]
```

15.149.3.5 popTop()

```
template<typename T , typename Compare = std::less<T>>
T smtrat::DynamicPriorityQueue< T, Compare >::popTop ( )  [inline]
```

15.149.3.6 push() [1/2]

```
template<typename T , typename Compare = std::less<T>>
void smtrat::DynamicPriorityQueue< T, Compare >::push (
            const T & t )  [inline]
```

15.149.3.7 push() [2/2]

```
template<typename T , typename Compare = std::less<T>>
void smtrat::DynamicPriorityQueue< T, Compare >::push (
            T && t )  [inline]
```

15.149.3.8 size()

```
template<typename T , typename Compare = std::less<T>>
std::size_t smtrat::DynamicPriorityQueue< T, Compare >::size ( ) const  [inline]
```

15.149.3.9 top()

```
template<typename T , typename Compare = std::less<T>>
const T& smtrat::DynamicPriorityQueue< T, Compare >::top ( ) const  [inline]
```

15.149.4 Field Documentation

15.149.4.1 c

```
template<typename T , typename Compare = std::less<T>>
Sequence smtrat::DynamicPriorityQueue< T, Compare >::c  [protected]
```

15.149.4.2 comp

```
template<typename T , typename Compare = std::less<T>>
Compare smtrat::DynamicPriorityQueue< T, Compare >::comp  [protected]
```

15.150 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 T & reference
- typedef T * 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)

Data Fields

- const typedef T * const_iterator
- const typedef T & const_reference
- const typedef T * const_pointer

15.150.1 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.

15.150.2 Member Typedef Documentation

15.150.2.1 iterator `template<typename T>`

```
typedef T* smtrat::dynarray< T >::iterator
```

15.150.2.2 pointer `template<typename T>`

```
typedef T* smtrat::dynarray< T >::pointer
```

15.150.2.3 reference `template<typename T>`

```
typedef T& smtrat::dynarray< T >::reference
```

15.150.2.4 value_type `template<typename T>`

```
typedef T smtrat::dynarray< T >::value_type
```

15.150.3 Constructor & Destructor Documentation

15.150.3.1 dynarray() [1/4] `template<typename T>`

```
smtrat::dynarray< T >::dynarray ( ) [inline]
```

15.150.3.2 dynarray() [2/4] `template<typename T>`

```
smtrat::dynarray< T >::dynarray (
            std::size_t initialCap ) [inline]
```

15.150.3.3 ~dynarray() `template<typename T>`

```
smtrat::dynarray< T >::~dynarray ( ) [inline]
```

15.150.3.4 dynarray() [3/4] template<typename T>

```
smtrat::dynarray< T >::dynarray (
            const dynarray< T > &  )  [delete]
```

15.150.3.5 dynarray() [4/4] template<typename T>

```
smtrat::dynarray< T >::dynarray (
            dynarray< T > && other )  [inline]
```

15.150.4 Member Function Documentation

15.150.4.1 back() template<typename T>

```
T& smtrat::dynarray< T >::back ( )  [inline]
```

15.150.4.2 begin() [1/2] template<typename T>

```
const_iterator smtrat::dynarray< T >::begin ( ) const  [inline]
```

15.150.4.3 begin() [2/2] template<typename T>

```
iterator smtrat::dynarray< T >::begin ( )  [inline]
```

15.150.4.4 capacity() template<typename T>

```
std::size_t smtrat::dynarray< T >::capacity ( ) const  [inline]
```

15.150.4.5 clear() template<typename T>

```
void smtrat::dynarray< T >::clear ( )  [inline]
```

15.150.4.6 consume() template<typename T>

```
void smtrat::dynarray< T >::consume (
            dynarray< T > && other )  [inline]
```

15.150.4.7 emplace_back() template<typename T>

```
template<typename...  Args>
T& smtrat::dynarray< T >::emplace_back (
            Args &&...  args )  [inline]
```

15.150.4.8 empty() template<typename T>

```
bool smtrat::dynarray< T >::empty ( ) const  [inline]
```

15.150.4.9 end() [1/2] template<typename T>

```
const_iterator smtrat::dynarray< T >::end ( ) const  [inline]
```

15.150.4.10 end() [2/2]

```
template<typename T>
iterator smtrat::dynarray< T >::end ( ) [inline]
```

15.150.4.11 front()

```
template<typename T>
T& smtrat::dynarray< T >::front ( ) [inline]
```

15.150.4.12 operator=() [1/2]

```
template<typename T>
dynarray& smtrat::dynarray< T >::operator= (
            const dynarray< T > & ) [delete]
```

15.150.4.13 operator=() [2/2]

```
template<typename T>
dynarray& smtrat::dynarray< T >::operator= (
            dynarray< T > && other ) [inline]
```

15.150.4.14 operator[]() [1/2]

```
template<typename T>
const_reference smtrat::dynarray< T >::operator[] (
            std::size_t index ) const [inline]
```

15.150.4.15 operator[]() [2/2]

```
template<typename T>
reference smtrat::dynarray< T >::operator[] (
            std::size_t index ) [inline]
```

15.150.4.16 pop_back()

```
template<typename T>
void smtrat::dynarray< T >::pop_back ( ) [inline]
```

15.150.4.17 remove()

```
template<typename T>
void smtrat::dynarray< T >::remove (
            const T & value ) [inline]
```

15.150.4.18 reserve()

```
template<typename T>
void smtrat::dynarray< T >::reserve (
            std::size_t minCap ) [inline]
```

15.150.4.19 size()

```
template<typename T>
std::size_t smtrat::dynarray< T >::size ( ) const [inline]
```

15.150.4.20 swap() [1/2] template<typename T>

```
void smtrat::dynarray< T >::swap (
            dynarray< T > && other )  [inline]
```

15.150.4.21 swap() [2/2] template<typename T>

```
void smtrat::dynarray< T >::swap (
            dynarray< T > & other )  [inline]
```

15.150.5 Field Documentation

15.150.5.1 const_iterator template<typename T>

```
const typedef T* smtrat::dynarray< T >::const_iterator
```

15.150.5.2 const_pointer template<typename T>

```
const typedef T* smtrat::dynarray< T >::const_pointer
```

15.150.5.3 const_reference template<typename T>

```
const typedef T& smtrat::dynarray< T >::const_reference
```

15.151 smtrat::dynarray_allocator< T > Class Template Reference

```
#include <alloc.h>
```

Friends

- class dynarray< T >

15.151.1 Friends And Related Function Documentation

15.151.1.1 dynarray< T > template<typename T>

```
friend class dynarray< T >  [friend]
```

15.152 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 isEmpty () 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 ()

15.152.1 Constructor & Destructor Documentation

15.152.1.1 DynHeap() [1/2] template<typename KeyType , class Compare = less<KeyType>, typename WeightType = std::size_t>

```
smtrat::datastructures::DynHeap< KeyType, Compare, WeightType >::DynHeap ( )  [inline], [explicit]
```

15.152.1.2 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]
```

15.152.2 Member Function Documentation

15.152.2.1 clear()

```
template<typename KeyType , class Compare = less<KeyType>, typename Weight↩
Type = std::size_t>
void smtrat::datastructures::DynHeap< KeyType, Compare, WeightType >::clear ( )  [inline]
```

15.152.2.2 extractMin()

```
template<typename KeyType , class Compare = less<KeyType>, typename
WeightType = std::size_t>
KeyType smtrat::datastructures::DynHeap< KeyType, Compare, WeightType >::extractMin ( )  [inline]
```

15.152.2.3 insert()

```
template<typename KeyType , class Compare = less<KeyType>, typename Weight↩
Type = std::size_t>
void smtrat::datastructures::DynHeap< KeyType, Compare, WeightType >::insert (
            const KeyType & pKey,
            const WeightType pWeight )  [inline]
```

15.152.2.4 isEmpty()

```
template<typename KeyType , class Compare = less<KeyType>, typename
WeightType = std::size_t>
bool smtrat::datastructures::DynHeap< KeyType, Compare, WeightType >::isEmpty ( ) const  [inline]
```

15.152.2.5 size()

```
template<typename KeyType , class Compare = less<KeyType>, typename Weight↩
Type = std::size_t>
std::size_t smtrat::datastructures::DynHeap< KeyType, Compare, WeightType >::size ( ) const
[inline]
```

15.152.2.6 update()

```
template<typename KeyType , class Compare = less<KeyType>, typename Weight↩
Type = std::size_t>
void smtrat::datastructures::DynHeap< KeyType, Compare, WeightType >::update (
            const KeyType & pKey,
            const WeightType pWeight )  [inline]
```

15.153 smtrat::cad::ProjectionGlobalInformation::ECData Struct Reference

```
#include <ProjectionInformation.h>
```

Data Fields

- std::size_t level = 0
- carl::Bitset polynomials

15.153.1 Field Documentation

15.153.1.1 level `std::size_t smtrat::cad::ProjectionGlobalInformation::ECData::level = 0`

15.153.1.2 polynomials `carl::Bitset smtrat::cad::ProjectionGlobalInformation::ECData::polynomials`

15.154 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

15.154.1 Detailed Description

template<typename VertexProperty, typename EdgeProperty, bool UniqueVertices = true>
struct smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Edge

Internal type of an edge.

15.154.2 Constructor & Destructor Documentation

15.154.2.1 Edge()

```
template<typename VertexProperty, typename EdgeProperty, bool UniqueVertices
= true>
smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Edge::Edge (
            const EdgeProperty & ep )  [inline]
```

Constructor from a property.

15.154.3 Member Function Documentation

15.154.3.1 addOut()

```
template<typename VertexProperty, typename EdgeProperty, bool Unique↩
Vertices = true>
void smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Edge::addOut (
            Vertex v )  [inline]
```

Add v to the outgoing vertices of this edge.

15.154.3.2 operator*() [1/2]

```
template<typename VertexProperty, typename EdgeProperty, bool
UniqueVertices = true>
EdgeProperty& smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >↩
::Edge::operator* ( )  [inline]
```

Access the property.

15.154.3.3 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.

15.154.3.4 operator->() [1/2]

```
template<typename VertexProperty, typename EdgeProperty, bool
UniqueVertices = true>
EdgeProperty* smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >↩
::Edge::operator-> ( )  [inline]
```

Access the property.

15.154.3.5 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.

15.154.3.6 out()

```
template<typename VertexProperty, typename EdgeProperty, bool UniqueVertices
= true>
const std::vector<Vertex>& smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, Unique↩
Vertices >::Edge::out ( ) const  [inline]
```

Retrieve the list of outgoing vertices.

15.154.4 Field Documentation

15.154.4.1 ForwardHyperGraph

```
template<typename VertexProperty, typename EdgeProperty, bool
UniqueVertices = true>
friend smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Edge::↩
ForwardHyperGraph
```

15.155 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

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)

Data Fields

- const typedef EdgeType *const * const_iterator

Friends

- class graph_info

15.155.1 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.

15.155.2 Member Typedef Documentation

15.155.2.1 iterator

```
template<typename Settings>
template<typename EdgeType>
typedef EdgeType** smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >↩
::iterator
```

15.155.3 Constructor & Destructor Documentation

15.155.3.1 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]
```

15.155.3.2 ~edge_list_type()

```
template<typename Settings>
template<typename EdgeType>
smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >::~edge_list_type ( )
```

15.155.4 Member Function Documentation

15.155.4.1 add()

```
template<typename Settings>
template<typename EdgeType>
void smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >::add (
            EdgeType * edge ) [inline]
```

15.155.4.2 begin() [1/2]

```
template<typename Settings>
template<typename EdgeType>
const_iterator smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >::begin ( )
const [inline], [noexcept]
```

15.155.4.3 begin() [2/2]

```
template<typename Settings>
template<typename EdgeType>
iterator smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >::begin ( ) [inline],
[noexcept]
```

15.155.4.4 end() [1/2]

```
template<typename Settings>
template<typename EdgeType>
const_iterator smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >::end ( )
const [inline], [noexcept]
```

15.155.4.5 end() [2/2]

```
template<typename Settings>
template<typename EdgeType>
iterator smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >::end ( ) [inline],
[noexcept]
```

15.155.4.6 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]
```

15.155.4.7 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]
```

15.155.4.8 remove() template<typename Settings>

```
template<typename EdgeType>
void smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >::remove (
            EdgeType * edge )  [inline]
```

15.155.4.9 remove_back() template<typename Settings>

```
template<typename EdgeType>
void smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >::remove_back ( )
[inline]
```

15.155.4.10 remove_destroy() template<typename Settings>

```
template<typename EdgeType>
void smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >::remove_destroy (
            EdgeType * edge )  [inline]
```

15.155.4.11 size() template<typename Settings>

```
template<typename EdgeType>
std::size_t smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >::size ( )
const  [inline], [noexcept]
```

15.155.5 Friends And Related Function Documentation

15.155.5.1 graph_info template<typename Settings>

```
template<typename EdgeType>
friend class graph_info  [friend]
```

15.155.6 Field Documentation

15.155.6.1 const_iterator template<typename Settings>

```
template<typename EdgeType>
const typedef EdgeType* const* smtrat::EQModule< Settings >::graph_info::edge_list_type< Edge↩
Type >::const_iterator
```

15.156 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 ()

15.156.1 Constructor & Destructor Documentation

15.156.1.1 ElementWithOrigins() [1/2] template<typename Element , typename Origin >

```
smtrat::ElementWithOrigins< Element, Origin >::ElementWithOrigins (
            const Element & _element )  [inline]
```

15.156.1.2 ElementWithOrigins() [2/2]

```
template<typename Element , typename Origin >
smtrat::ElementWithOrigins< Element, Origin >::ElementWithOrigins (
            const Element & _element,
            const Origin & _origin ) [inline]
```

15.156.2 Member Function Documentation

15.156.2.1 addOrigin()

```
template<typename Element , typename Origin >
void smtrat::ElementWithOrigins< Element, Origin >::addOrigin (
            const Origin & _origin ) [inline]
```

15.156.2.2 element()

```
template<typename Element , typename Origin >
const Element& smtrat::ElementWithOrigins< Element, Origin >::element ( ) const [inline]
```

15.156.2.3 hasOrigins()

```
template<typename Element , typename Origin >
bool smtrat::ElementWithOrigins< Element, Origin >::hasOrigins ( ) [inline]
```

15.156.2.4 origins()

```
template<typename Element , typename Origin >
const std::set<Origin>& smtrat::ElementWithOrigins< Element, Origin >::origins ( ) const
[inline]
```

15.156.2.5 removeOrigin()

```
template<typename Element , typename Origin >
bool smtrat::ElementWithOrigins< Element, Origin >::removeOrigin (
            const Origin & _origin ) [inline]
```

15.157 smtrat::EMModule< Settings > Class Template Reference

```
#include <EMModule.h>
```

Public Types

- typedef Settings SettingsType
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::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 smtrat::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 _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.
- 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 &_branching↩Value) 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 _preferLeft↩Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩::vector< FormulaT >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩CaseWeak=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.

15.157.1 Member Typedef Documentation

15.157.1.1 SettingsType `template<typename Settings >`

```
typedef Settings smtrat::EMModule< Settings >::SettingsType
```

15.157.2 Member Enumeration Documentation

15.157.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

15.157.3 Constructor & Destructor Documentation

15.157.3.1 EMModule() `template<class Settings >`

```
smtrat::EMModule< Settings >::EMModule (
```

```
const ModuleInput * _formula,
Conditionals & _conditionals,
Manager * _manager = NULL )
```

15.157.3.2 ~EMModule() template<class Settings >

smtrat::EMModule< Settings >::~EMModule ( )

15.157.4 Member Function Documentation

15.157.4.1 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

_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.

15.157.4.2 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 >.

15.157.4.3 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

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 smtrat::SATModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::VSModule< Settings >, smtrat::PFEModule< Settings >, smtrat::EQModule< Setting

smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Sett smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, and smtrat::GBModule< Settings >.

15.157.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.157.4.5 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.

15.157.4.6 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.

15.157.4.7 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.

15.157.4.8 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.

15.157.4.9 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.

15.157.4.10 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

_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.

15.157.4.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns All satisfying assignments, if existent.

15.157.4.12 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.

15.157.4.13 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.

15.157.4.14 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.

15.157.4.15 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.

15.157.4.16 branchAt() [1/6]

```
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]
```

15.157.4.17 branchAt() [2/6]

```
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]
```

15.157.4.18 branchAt() [3/6]

```
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]
```

15.157.4.19 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.

Parameters

_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.157.4.20 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            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]
```

15.157.4.21 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

15.157.4.22 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

15.157.4.23 checkCore()

```
template<class 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.

15.157.4.24 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
    std::vector< FormulaSetT >::const_iterator _infsubset,
    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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.157.4.25 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.

15.157.4.26 cleanModel()

```
void smtrat::Module::cleanModel ( ) const [inline], [protected],
[inherited]
```

Substitutes variable occurrences by its model value in the model values of other variables.

15.157.4.27 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.157.4.28 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.157.4.29 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.157.4.30 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.157.4.31 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

15.157.4.32 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.

15.157.4.33 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.157.4.34 constraintsToInform() `const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ToInform ( ) const [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.157.4.35 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

15.157.4.36 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)

15.157.4.37 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

_constraint	The constraint to remove from internal data structures.

15.157.4.38 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.157.4.39 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.157.4.40 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.157.4.41 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.157.4.42 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
            const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.157.4.43 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.

15.157.4.44 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.

15.157.4.45 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

15.157.4.46 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

15.157.4.47 getBackendsAllModels()

`void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

15.157.4.48 getBackendsModel()

`void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

15.157.4.49 getInfeasibleSubsets() [1/2]

`void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

15.157.4.50 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.157.4.51 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.157.4.52 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.

15.157.4.53 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.157.4.54 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.157.4.55 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.157.4.56 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 reveived formula, if the Boolean is true.

Reimplemented from smtrat::Module.

15.157.4.57 hasLemmas()

`bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

15.157.4.58 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).

15.157.4.59 id()

`std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns A unique ID to identify this module instance.

15.157.4.60 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.

15.157.4.61 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.157.4.62 informBackends()

```
void smtrat::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.

15.157.4.63 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, and smtrat::ICPModule< ICPSettings1 >.

15.157.4.64 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.157.4.65 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::FPPModule< Settings >.

15.157.4.66 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.157.4.67 isLemmaLevel() `bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.157.4.68 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().

15.157.4.69 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.

15.157.4.70 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.157.4.71 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.157.4.72 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.

15.157.4.73 moduleName() `virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]`

Returns The name of the given module type as name.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::EQPreprocessingModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::VSModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::GBModule< Settings >, smtrat::IntEqModule< Setti smtrat::CubeLIAModule< Settings >, smtrat::CurryModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IncWidthModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::BEModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::SplitSOSModule< Settings >.

15.157.4.74 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

15.157.4.75 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

15.157.4.76 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula. TODO: disable this method

15.157.4.77 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula.

15.157.4.78 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to the backends of this module.

15.157.4.79 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to this module.

15.157.4.80 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.157.4.81 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

_out	The stream to print the assignment on.

15.157.4.82 printInfeasibleSubsets()

`void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.157.4.83 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

_out	The stream to print the assignment on.

15.157.4.84 printPassedFormula()

`void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.157.4.85 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.157.4.86 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.

15.157.4.87 receivedFormulaChecked()

```
void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]
```

Notifies that the received formulas has been checked.

15.157.4.88 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩InfeasibleSubset (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

15.157.4.89 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const [inline], [inherited]
```

15.157.4.90 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

_subformula	The sub formula of the received formula to remove.

Reimplemented from smtrat::Module.

15.157.4.91 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

formula	The sub formula of the received formula to remove.

Reimplemented in smtrat::SATModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::VSModule< Settings >, smtrat::PFEModule< Settings >, smtrat::EQModule< Setting smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Sett smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, and smtrat::GBModule< Settings >.

15.157.4.92 removeOrigin()

`std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin ( ModuleInput::iterator _formula, const FormulaT & _origin ) [inline], [protected], [inherited]`

15.157.4.93 removeOrigins()

`std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩ Origins ( ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]`

15.157.4.94 rPassedFormula()

`const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]`

Returns A reference to the formula passed to the backends of this module.

15.157.4.95 rReceivedFormula()

`const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]`

Returns A reference to the formula passed to this module.

15.157.4.96 runBackends() [1/2]

`virtual Answer smtrat::PModule::runBackends ( ) [inline], [virtual], [inherited]`

Reimplemented from smtrat::Module.

15.157.4.97 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.

15.157.4.98 setId()

```
void smtrat::Module::setId (
    std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.157.4.99 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.

15.157.4.100 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.

15.157.4.101 splitUnequalConstraint()

`void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.157.4.102 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.157.4.103 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 >.

15.157.4.104 updateLemmas()

`void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.157.4.105 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.

15.157.4.106 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).

15.157.5 Field Documentation

15.157.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.157.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.157.5.3 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.

15.157.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.157.5.5 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

15.157.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.157.5.7 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.

15.157.5.8 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.

15.157.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.157.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.157.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.157.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.157.5.13 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.

15.157.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.157.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.157.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.157.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.157.5.18 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.

15.157.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]`

Reusable splitting variables.

15.157.5.20 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.157.5.21 mSmallerMusesCheckCounter

`unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.157.5.22 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.

15.157.5.23 mTheoryPropagations

`std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]`

15.157.5.24 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).

15.157.5.25 mVariableCounters

`std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.158 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

15.158.1 Member Typedef Documentation

15.158.1.1 Graph

```
template<typename VertexName>
using smtrat::EQGraph< VertexName >::Graph = boost::adjacency_list<vec, vec, undirected,
vertex_property>
```

15.158.1.2 MaybePath

```
template<typename VertexName>
using smtrat::EQGraph< VertexName >::MaybePath = std::optional<std::vector<PathEdge> >
```

15.158.1.3 PathEdge

```
template<typename VertexName>
using smtrat::EQGraph< VertexName >::PathEdge = std::pair<VertexName, VertexName>
```

15.158.1.4 undirected

```
template<typename VertexName>
using smtrat::EQGraph< VertexName >::undirected = boost::undirectedS
```

15.158.1.5 vec

```
template<typename VertexName>
using smtrat::EQGraph< VertexName >::vec = boost::vecS
```

15.158.1.6 Vertex

```
template<typename VertexName>
using smtrat::EQGraph< VertexName >::Vertex = typename boost::graph_traits<Graph>::vertex_↩
descriptor
```

15.158.1.7 vertex_property

```
template<typename VertexName>
using smtrat::EQGraph< VertexName >::vertex_property = boost::property<vertex_properties_t,
VertexName>
```

15.158.2 Member Function Documentation

15.158.2.1 add_edge()

```
template<typename VertexName>
void smtrat::EQGraph< VertexName >::add_edge (
            VertexName const & a,
            VertexName const & b )  [inline], [noexcept]
```

15.158.2.2 add_vertex()

```
template<typename VertexName>
void smtrat::EQGraph< VertexName >::add_vertex (
            VertexName const & name )  [inline], [noexcept]
```

15.158.2.3 get_path() [1/2]

```
template<typename VertexName>
MaybePath smtrat::EQGraph< VertexName >::get_path (
            VertexName const & a,
            VertexName const & b )  [inline], [noexcept]
```

15.158.2.4 get_path() [2/2]

```
template<typename VertexName>
MaybePath smtrat::EQGraph< VertexName >::get_path (
            VertexName const & a,
            VertexName const & b,
            size_t bound )  [inline], [noexcept]
```

15.158.2.5 properties() [1/4]

```
template<typename VertexName>
auto smtrat::EQGraph< VertexName >::properties ( ) const  [inline], [noexcept]
```

15.158.2.6 properties() [2/4]

```
template<typename VertexName>
auto smtrat::EQGraph< VertexName >::properties ( )  [inline], [noexcept]
```

15.158.2.7 properties() [3/4]

```
template<typename VertexName>
VertexName const& smtrat::EQGraph< VertexName >::properties (
            Vertex const & v ) const  [inline], [noexcept]
```

15.158.2.8 properties() [4/4]

```
template<typename VertexName>
VertexName& smtrat::EQGraph< VertexName >::properties (
            Vertex const & v )  [inline], [noexcept]
```

15.158.2.9 remove_edge()

```
template<typename VertexName>
void smtrat::EQGraph< VertexName >::remove_edge (
            VertexName const & a,
            VertexName const & b )  [inline], [noexcept]
```

15.158.3 Field Documentation

15.158.3.1 graph `template<typename VertexName>`

`Graph smtrat::EQGraph< VertexName >::graph`

15.158.3.2 vertices `template<typename VertexName>`

`std::map<VertexName, Vertex> smtrat::EQGraph< VertexName >::vertices`

15.159 smtrat::EQModule< Settings > Class Template Reference

`#include <EQModule.h>`

Public Types

- typedef Settings SettingsType
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- EQModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=NULL)

  Constructors.
- ~EQModule ()

  Destructor.
- 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 * find_representative (const term_type &term)

  find a "canonical" representative for some term
- std::size_t get_class (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 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 _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 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 _preferLeft↩ Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩ ::vector< FormulaT >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > && _premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩ AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩ CaseWeak=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.

15.159.1 Member Typedef Documentation

15.159.1.1 SettingsType `template<typename Settings>`

```
typedef Settings smtrat::EQModule< Settings >::SettingsType
```

15.159.2 Member Enumeration Documentation

15.159.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

15.159.3 Constructor & Destructor Documentation

15.159.3.1 EQModule() `template<class Settings >`

```
smtrat::EQModule< Settings >::EQModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = NULL )
```

Constructors.

15.159.3.2 ~EQModule() `template<class Settings >`

```
smtrat::EQModule< Settings >::~EQModule ( )
```

Destructor.

15.159.4 Member Function Documentation

15.159.4.1 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.

15.159.4.2 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 >.

15.159.4.3 addCore()

```
template<class 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.

15.159.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.159.4.5 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.

15.159.4.6 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.

15.159.4.7 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.

15.159.4.8 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.

15.159.4.9 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.

15.159.4.10 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

_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.

15.159.4.11 allModels()

`const std::vector<Model>& smtrat::Module::allModels ( ) const [inline], [inherited]`

Returns All satisfying assignments, if existent.

15.159.4.12 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.

15.159.4.13 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.

15.159.4.14 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.

15.159.4.15 branchAt() [1/6]

```
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]
```

15.159.4.16 branchAt() [2/6]

```
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]
```

15.159.4.17 branchAt() [3/6]

```
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]
```

15.159.4.18 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.159.4.19 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
    const typename P::PolyType & _poly,
    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]
```

15.159.4.20 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false )  [inline], [protected], [inherited]
```

15.159.4.21 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in smtrat::PModule.

15.159.4.22 checkCore()

```
template<class 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.

15.159.4.23 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
            std::vector< FormulaSetT >::const_iterator _infsubset,
            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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.159.4.24 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.

15.159.4.25 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.159.4.26 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.159.4.27 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.159.4.28 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.159.4.29 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.159.4.30 collectOrigins() [1/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

15.159.4.31 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

_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.

15.159.4.32 collectTheoryPropagations()

`void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.159.4.33 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.159.4.34 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

15.159.4.35 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)

15.159.4.36 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

_constraint	The constraint to remove from internal data structures.

15.159.4.37 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.159.4.38 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.159.4.39 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.159.4.40 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.159.4.41 find_representative()

```
template<typename Settings >
auto smtrat::EQModule< Settings >::find_representative (
            const term_type & term )
```

find a "canonical" representative for some term

15.159.4.42 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBest↩
Origin (
            const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.159.4.43 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.

15.159.4.44 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.

15.159.4.45 freeSplittingVariable()

```
static void smtrat::Module::freeSplittingVariable (
            const FormulaT & _splittingVariable ) [inline], [static], [inherited]
```

15.159.4.46 generateTrivialInfeasibleSubset()

```
void smtrat::Module::generateTrivialInfeasibleSubset
( ) [inline], [protected], [inherited]
```

Stores the trivial infeasible subset being the set of received formulas.

15.159.4.47 get_class()

```
template<typename Settings >
std::size_t smtrat::EQModule< Settings >::get_class (
            const term_type & term )
```

returns the component-component-level class of some term

15.159.4.48 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

15.159.4.49 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

15.159.4.50 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

15.159.4.51 getInfeasibleSubsets() [2/2] `std::vector< FormulaSetT > smtrat::Module::getInfeasible↩ Subsets ( 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.159.4.52 getInformationRelevantFormulas() `const std::set< FormulaT > & smtrat::Module::get↩ InformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.159.4.53 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.

15.159.4.54 getOrigins() [1/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

15.159.4.55 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.159.4.56 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.159.4.57 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 reveived formula, if the Boolean is true.

Reimplemented in smtrat::PModule.

15.159.4.58 hasLemmas()

```
bool smtrat::Module::hasLemmas ( ) [inline], [inherited]
```

Checks whether this module or any of its backends provides any lemmas.

15.159.4.59 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).

15.159.4.60 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns A unique ID to identify this module instance.

15.159.4.61 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.

15.159.4.62 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.159.4.63 informBackends() `void smtrat::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.

15.159.4.64 informCore()

```
template<class Settings >
bool smtrat::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

_constraint	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.

15.159.4.65 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const  [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.159.4.66 init()

```
template<class Settings >
void smtrat::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 smtrat::Module.

15.159.4.67 is_minimizing()

```
bool smtrat::Module::is_minimizing ( ) const  [inline], [inherited]
```

15.159.4.68 isLemmaLevel()

```
bool smtrat::Module::isLemmaLevel (
            LemmaLevel level )  [protected], [inherited]
```

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.159.4.69 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().

15.159.4.70 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.

15.159.4.71 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.159.4.72 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.159.4.73 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

_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.

15.159.4.74 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.

15.159.4.75 objective()

```
carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]
```

15.159.4.76 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
    const FormulaT & _origin ) const [protected], [inherited]
```

15.159.4.77 passedFormulaBegin()

```
ModuleInput::iterator smtrat::Module::passedFormulaBegin ( )
[inline], [protected], [inherited]
```

Returns An iterator to the end of the passed formula. TODO: disable this method

15.159.4.78 passedFormulaEnd()

```
ModuleInput::iterator smtrat::Module::passedFormulaEnd ( )
[inline], [protected], [inherited]
```

Returns An iterator to the end of the passed formula.

15.159.4.79 pPassedFormula()

```
const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline],
[inherited]
```

Returns A pointer to the formula passed to the backends of this module.

15.159.4.80 pReceivedFormula()

```
const ModuleInput* smtrat::Module::pReceivedFormula ( ) const
[inline], [inherited]
```

Returns A pointer to the formula passed to this module.

15.159.4.81 print()

```
void smtrat::Module::print (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.159.4.82 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

_out	The stream to print the assignment on.

15.159.4.83 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.159.4.84 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

_out	The stream to print the assignment on.

15.159.4.85 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.159.4.86 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.159.4.87 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

_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.

15.159.4.88 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

15.159.4.89 receivedFormulaChecked()

```
void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]
```

Notifies that the received formulas has been checked.

15.159.4.90 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩InfeasibleSubset (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

15.159.4.91 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const [inline], [inherited]
```

15.159.4.92 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.

15.159.4.93 removeCore()

```
template<class 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

_subformula	The position of the subformula to remove.

Reimplemented from smtrat::Module.

15.159.4.94 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin
(
            ModuleInput::iterator _formula,
            const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.159.4.95 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

15.159.4.96 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline],
[inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.159.4.97 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.159.4.98 runBackends() [1/2]

`virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]`

Reimplemented in smtrat::PModule.

15.159.4.99 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

_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 in smtrat::PModule.

15.159.4.100 setId()

```
void smtrat::Module::setId (
    std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.159.4.101 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.

15.159.4.102 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.

15.159.4.103 splitUnequalConstraint()

`void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.159.4.104 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.159.4.105 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 >.

15.159.4.106 updateLemmas()

`void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.159.4.107 updateModel()

`template<class 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.

15.159.4.108 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).

15.159.5 Field Documentation

15.159.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.159.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.159.5.3 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.

15.159.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.159.5.5 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

15.159.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.159.5.7 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.

15.159.5.8 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.

15.159.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.159.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.159.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.159.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.159.5.13 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.

15.159.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.159.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.159.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.159.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.159.5.18 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.

15.159.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]`

Reusable splitting variables.

15.159.5.20 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.159.5.21 mSmallerMusesCheckCounter

`unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.159.5.22 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.

15.159.5.23 mTheoryPropagations

`std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]`

15.159.5.24 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).

15.159.5.25 mVariableCounters

`std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.160 smtrat::EQPreprocessingModule< Settings > Class Template Reference

`#include <EQPreprocessingModule.h>`

Public Types

- typedef Settings SettingsType
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- EQPreprocessingModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=NULL)

  Constructors.
- ~EQPreprocessingModule ()

  Destructor.
- 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 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 _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.

- 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 &_branching↩Value) 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 _preferLeft↩Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩::vector< FormulaT >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩CaseWeak=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.

15.160.1 Member Typedef Documentation

15.160.1.1 SettingsType

```
template<typename Settings >
typedef Settings smtrat::EQPreprocessingModule< Settings >::SettingsType
```

15.160.2 Member Enumeration Documentation

15.160.2.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

15.160.3 Constructor & Destructor Documentation

15.160.3.1 EQPreprocessingModule()

```
template<class Settings >
smtrat::EQPreprocessingModule< Settings >::EQPreprocessingModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = NULL )
```

Constructors.

15.160.3.2 ~EQPreprocessingModule()

```
template<class Settings >
smtrat::EQPreprocessingModule< Settings >::~EQPreprocessingModule ( )
```

Destructor.

15.160.4 Member Function Documentation

15.160.4.1 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.

15.160.4.2 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 >.

15.160.4.3 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

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 smtrat::SATModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::VSModule< Settings >, smtrat::PFEModule< Settings >, smtrat::EQModule< Setting smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Sett smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, and smtrat::GBModule< Settings >.

15.160.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.160.4.5 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.

15.160.4.6 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.

15.160.4.7 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.

15.160.4.8 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.

15.160.4.9 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.

15.160.4.10 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

_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.

15.160.4.11 allModels() `const std::vector<Model>& smtrat::Module::allModels ( ) const [inline], [inherited]`

Returns All satisfying assignments, if existent.

15.160.4.12 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.

15.160.4.13 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.

15.160.4.14 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.

15.160.4.15 branchAt() [1/6]

```
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]
```

15.160.4.16 branchAt() [2/6]

```
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]
```

15.160.4.17 branchAt() [3/6]

```
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]
```

15.160.4.18 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.160.4.19 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
    const typename P::PolyType & _poly,
    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]
```

15.160.4.20 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false )  [inline], [protected], [inherited]
```

15.160.4.21 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in smtrat::PModule.

15.160.4.22 checkCore()

```
template<class 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.

15.160.4.23 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
            std::vector< FormulaSetT >::const_iterator _infsubset,
            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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.160.4.24 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.

15.160.4.25 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.160.4.26 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.160.4.27 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.160.4.28 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.160.4.29 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.160.4.30 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

15.160.4.31 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

_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.

15.160.4.32 collectTheoryPropagations()

`void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.160.4.33 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ToInform ( ) const [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.160.4.34 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

15.160.4.35 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)

15.160.4.36 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

_constraint	The constraint to remove from internal data structures.

15.160.4.37 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.160.4.38 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.160.4.39 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.160.4.40 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.160.4.41 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.160.4.42 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.

15.160.4.43 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.

15.160.4.44 freeSplittingVariable()

```
static void smtrat::Module::freeSplittingVariable (
    const FormulaT & _splittingVariable ) [inline], [static], [inherited]
```

15.160.4.45 generateTrivialInfeasibleSubset()

```
void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]
```

Stores the trivial infeasible subset being the set of received formulas.

15.160.4.46 getBackendsAllModels()

```
void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]
```

Stores all models of a backend in the list of all models of this module.

15.160.4.47 getBackendsModel()

```
void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]
```

15.160.4.48 getInfeasibleSubsets() [1/2]

```
void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]
```

Copies the infeasible subsets of the passed formula.

15.160.4.49 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.160.4.50 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.160.4.51 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.

15.160.4.52 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.160.4.53 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.160.4.54 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.160.4.55 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 reveived formula, if the Boolean is true.

Reimplemented in smtrat::PModule.

15.160.4.56 hasLemmas()

`bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

15.160.4.57 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).

15.160.4.58 id()

`std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns A unique ID to identify this module instance.

15.160.4.59 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.

15.160.4.60 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.160.4.61 informBackends()

`void smtrat::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.

15.160.4.62 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, and smtrat::ICPModule< ICPSettings1 >.

15.160.4.63 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.

15.160.4.64 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::FPPModule< Settings >.

15.160.4.65 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.160.4.66 isLemmaLevel() `bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.160.4.67 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().

15.160.4.68 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.

15.160.4.69 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.160.4.70 model()

```
const Model& smtrat::Module::model ( ) const [inline], [inherited]
```

Returns The assignment of the current satisfiable result, if existent.

15.160.4.71 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

_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.

15.160.4.72 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.

15.160.4.73 objective()

```
carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]
```

15.160.4.74 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
    const FormulaT & _origin ) const [protected], [inherited]
```

15.160.4.75 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula. TODO: disable this method

15.160.4.76 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula.

15.160.4.77 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to the backends of this module.

15.160.4.78 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to this module.

15.160.4.79 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.160.4.80 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

_out	The stream to print the assignment on.

15.160.4.81 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const  [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.160.4.82 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

_out	The stream to print the assignment on.

15.160.4.83 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const  [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.160.4.84 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const  [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.160.4.85 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.

15.160.4.86 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

15.160.4.87 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

15.160.4.88 receivedVariable()

`bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

15.160.4.89 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

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

15.160.4.90 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

formula	The sub formula of the received formula to remove.

Reimplemented in smtrat::SATModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::VSModule< Settings >, smtrat::PFEModule< Settings >, smtrat::EQModule< Setting smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Sett smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, and smtrat::GBModule< Settings >.

15.160.4.91 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.160.4.92 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

15.160.4.93 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline],
[inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.160.4.94 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.160.4.95 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( ) [inline],
[protected], [virtual], [inherited]
```

Reimplemented in smtrat::PModule.

15.160.4.96 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

_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 in smtrat::PModule.

15.160.4.97 setId()

```
void smtrat::Module::setId ( std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.160.4.98 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.

15.160.4.99 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.

15.160.4.100 splitUnequalConstraint()

```
void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]
```

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.160.4.101 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.160.4.102 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 >.

15.160.4.103 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.160.4.104 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.

15.160.4.105 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).

15.160.5 Field Documentation

15.160.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.160.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.160.5.3 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.

15.160.5.4 mConstraintsToInform

`carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.160.5.5 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

15.160.5.6 mFirstPosInLastBranches

`std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.160.5.7 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.

15.160.5.8 mFirstUncheckedReceivedSubformula

`ModuleInput::const_iterator smtrat::Module::m↩FirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

15.160.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.160.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.160.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.160.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.160.5.13 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.

15.160.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.160.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.160.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.160.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.160.5.18 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.

15.160.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]`

Reusable splitting variables.

15.160.5.20 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.160.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.160.5.22 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.

15.160.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

15.160.5.24 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).

15.160.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.161 Minisat::Equal< K > Struct Template Reference

```
#include <Map.h>
```

Public Member Functions

- bool operator() (const K &k1, const K &k2) const

15.161.1 Member Function Documentation

15.161.1.1 operator()() `template<class K>`

```
bool Minisat::Equal< K >::operator() (
            const K & k1,
            const K & k2 ) const [inline]
```

15.162 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

15.162.1 Member Function Documentation

15.162.1.1 operator()() template<typename T >

```
qi::error_handler_result delta::ErrorHandler::operator() (
            T b,
            T e,
            T where ) const  [inline]
```

15.163 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

15.163.1 Member Function Documentation

15.163.1.1 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]
```

15.164 smtrat::ESModule< Settings > Class Template Reference

```
#include <ESModule.h>
```

Public Types

- typedef Settings SettingsType
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::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 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
- 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 _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.

- 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 _preferLeft↩ Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩ ::vector< FormulaT >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩ AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩ CaseWeak=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.

15.164.1 Member Typedef Documentation

15.164.1.1 SettingsType

```
template<typename Settings >
typedef Settings smtrat::ESModule< Settings >::SettingsType
```

15.164.2 Member Enumeration Documentation

15.164.2.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

15.164.3 Constructor & Destructor Documentation

15.164.3.1 ESModule()

```
template<class Settings >
smtrat::ESModule< Settings >::ESModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = NULL )
```

15.164.3.2 ~ESModule()

```
template<class Settings >
smtrat::ESModule< Settings >::~ESModule ( )
```

15.164.4 Member Function Documentation

15.164.4.1 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

_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.

15.164.4.2 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 >.

15.164.4.3 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

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 smtrat::SATModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::VSModule< Settings >, smtrat::PFEModule< Settings >, smtrat::EQModule< Setting smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Sett smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, and smtrat::GBModule< Settings >.

15.164.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.164.4.5 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.

15.164.4.6 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.

15.164.4.7 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.

15.164.4.8 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.

15.164.4.9 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.

15.164.4.10 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

_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.

15.164.4.11 allModels()

`const std::vector<Model>& smtrat::Module::allModels ( ) const [inline], [inherited]`

Returns All satisfying assignments, if existent.

15.164.4.12 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.

15.164.4.13 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.

15.164.4.14 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.

15.164.4.15 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.

15.164.4.16 branchAt() [1/6]

```
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]
```

15.164.4.17 branchAt() [2/6]

```
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]
```

15.164.4.18 branchAt() [3/6]

```
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]
```

15.164.4.19 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.164.4.20 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P
>> = carl::dummy>
bool smtrat::Module::branchAt (
    const typename P::PolyType & _poly,
```

```
            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]
```

15.164.4.21 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P
>> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false )  [inline], [protected], [inherited]
```

15.164.4.22 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

15.164.4.23 checkCore()

```
template<class 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.

15.164.4.24 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
    std::vector< FormulaSetT >::const_iterator _infsubset,
    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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.164.4.25 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.

15.164.4.26 cleanModel()

```
void smtrat::Module::cleanModel ( ) const [inline], [protected],
[inherited]
```

Substitutes variable occurrences by its model value in the model values of other variables.

15.164.4.27 clearLemmas()

```
void smtrat::Module::clearLemmas ( ) [inline], [inherited]
```

Deletes all yet found lemmas.

15.164.4.28 clearModel()

```
void smtrat::Module::clearModel ( ) const [inline], [protected],
[inherited]
```

Clears the assignment, if any was found.

15.164.4.29 clearModels()

```
void smtrat::Module::clearModels ( ) const [inline], [protected],
[inherited]
```

Clears all assignments, if any was found.

15.164.4.30 clearPassedFormula()

```
void smtrat::Module::clearPassedFormula ( ) [protected],
[inherited]
```

15.164.4.31 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

15.164.4.32 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.

15.164.4.33 collectTheoryPropagations()

```
void smtrat::Module::collectTheoryPropagations ( ) [inherited]
```

15.164.4.34 constraintsToInform()

```
const carl::FastSet<FormulaT>& smtrat::Module::constraints↩
ToInform ( ) const [inline], [inherited]
```

Returns The constraints which the backends must be informed about.

15.164.4.35 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

15.164.4.36 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)

15.164.4.37 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

_constraint	The constraint to remove from internal data structures.

15.164.4.38 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.164.4.39 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.164.4.40 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.164.4.41 excludeNotReceivedVariablesFromModel()

`void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

15.164.4.42 findBestOrigin()

`std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin ( const std::vector< FormulaT > & _origins ) const [protected], [inherited]`

Parameters

_origins	

Returns

15.164.4.43 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.

15.164.4.44 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.

15.164.4.45 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

15.164.4.46 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

15.164.4.47 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

15.164.4.48 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

15.164.4.49 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

15.164.4.50 getInfeasibleSubsets() [2/2] `std::vector< FormulaSetT > smtrat::Module::getInfeasible↩ Subsets ( 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.164.4.51 getInformationRelevantFormulas() `const std::set< FormulaT > & smtrat::Module::get↩ InformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.164.4.52 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.

15.164.4.53 getOrigins() [1/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

15.164.4.54 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.164.4.55 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.164.4.56 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 reveived formula, if the Boolean is true.

Reimplemented from smtrat::Module.

15.164.4.57 hasLemmas()

```
bool smtrat::Module::hasLemmas ( ) [inline], [inherited]
```

Checks whether this module or any of its backends provides any lemmas.

15.164.4.58 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).

15.164.4.59 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns A unique ID to identify this module instance.

15.164.4.60 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.

15.164.4.61 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.164.4.62 informBackends() `void smtrat::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.

15.164.4.63 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, and smtrat::ICPModule< ICPSettings1 >.

15.164.4.64 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.164.4.65 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::FPPModule< Settings >.

15.164.4.66 is_minimizing()

```
bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]
```

15.164.4.67 isLemmaLevel()

```
bool smtrat::Module::isLemmaLevel (
    LemmaLevel level ) [protected], [inherited]
```

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.164.4.68 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().

15.164.4.69 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.

15.164.4.70 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.164.4.71 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.164.4.72 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.

15.164.4.73 moduleName()

`virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]`

Returns The name of the given module type as name.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::EQPreprocessingModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::VSModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::GBModule< Settings >, smtrat::IntEqModule< Setti smtrat::CubeLIAModule< Settings >, smtrat::CurryModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IncWidthModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::BEModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::SplitSOSModule< Settings >.

15.164.4.74 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

15.164.4.75 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

15.164.4.76 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula. TODO: disable this method

15.164.4.77 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula.

15.164.4.78 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to the backends of this module.

15.164.4.79 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to this module.

15.164.4.80 print()

```
void smtrat::Module::print (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.164.4.81 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

_out	The stream to print the assignment on.

15.164.4.82 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.164.4.83 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

_out	The stream to print the assignment on.

15.164.4.84 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.164.4.85 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.164.4.86 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.

15.164.4.87 receivedFormulaChecked()

```
void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]
```

Notifies that the received formulas has been checked.

15.164.4.88 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩InfeasibleSubset (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

15.164.4.89 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const [inline], [inherited]
```

15.164.4.90 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

_subformula	The sub formula of the received formula to remove.

Reimplemented from smtrat::Module.

15.164.4.91 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

formula	The sub formula of the received formula to remove.

Reimplemented in smtrat::SATModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::VSModule< Settings >, smtrat::PFEModule< Settings >, smtrat::EQModule< Setting smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Sett smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, and smtrat::GBModule< Settings >.

15.164.4.92 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin
(
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.164.4.93 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

15.164.4.94 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline],
[inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.164.4.95 rReceivedFormula() `const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]`

Returns A reference to the formula passed to this module.

15.164.4.96 runBackends() [1/2] `virtual Answer smtrat::PModule::runBackends ( ) [inline], [virtual], [inherited]`

Reimplemented from smtrat::Module.

15.164.4.97 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.

15.164.4.98 setId() `void smtrat::Module::setId ( std::size_t _id ) [inline], [inherited]`

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.164.4.99 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.

15.164.4.100 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.

15.164.4.101 splitUnequalConstraint() `void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.164.4.102 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.164.4.103 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 >.

15.164.4.104 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.164.4.105 updateModel() `template<class 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.

15.164.4.106 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).

15.164.5 Field Documentation

15.164.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.164.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.164.5.3 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.

15.164.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.164.5.5 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

15.164.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.164.5.7 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.

15.164.5.8 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.

15.164.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.164.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.164.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.164.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.164.5.13 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.

15.164.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.164.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.164.5.16 mModelComputed `bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.164.5.17 mNumOfBranchVarsToStore `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.164.5.18 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.

15.164.5.19 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

15.164.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.164.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.164.5.22 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.

15.164.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

15.164.5.24 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).

15.164.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.165 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

15.165.1 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.

15.165.2 Member Function Documentation

15.165.2.1 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]
```

15.166 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

15.166.1 Member Function Documentation

15.166.1.1 estimate_distance()

```
template<typename IteratorType >
static constexpr std::size_t smtrat::uf_impl::estimator< IteratorType, false >::estimate_↩
distance (
            IteratorType ,
            IteratorType  )  [inline], [static], [constexpr], [noexcept]
```

15.167 smtrat::ExactlyOneCommanderEncoder Class Reference

```
#include <ExactlyOneCommanderEncoder.h>
```

Public Member Functions

- ExactlyOneCommanderEncoder ()
- Rational encodingSize (const ConstraintT &constraint)
- bool canEncode (const ConstraintT &constraint)
- std::string name ()
- boost::optional< FormulaT > encode (const ConstraintT &constraint)

  Encodes an arbitrary constraint.

Data Fields

- std::size_t problem_size

Protected Member Functions

- boost::optional< FormulaT > doEncode (const ConstraintT &constraint)
- FormulaT generateVarChain (const std::set< carl::Variable > &vars, carl::FormulaType type)

15.167.1 Constructor & Destructor Documentation

15.167.1.1 ExactlyOneCommanderEncoder() `smtrat::ExactlyOneCommanderEncoder::ExactlyOne↩CommanderEncoder ( ) [inline]`

15.167.2 Member Function Documentation

15.167.2.1 canEncode() `bool smtrat::ExactlyOneCommanderEncoder::canEncode ( const ConstraintT & constraint ) [virtual]`

Implements smtrat::PseudoBoolEncoder.

15.167.2.2 doEncode() `boost::optional< FormulaT > smtrat::ExactlyOneCommanderEncoder::do↩Encode ( const ConstraintT & constraint ) [protected], [virtual]`

Implements smtrat::PseudoBoolEncoder.

15.167.2.3 encode() `boost::optional< FormulaT > smtrat::PseudoBoolEncoder::encode ( const ConstraintT & constraint ) [inherited]`

Encodes an arbitrary constraint.

Returns encoded formula

15.167.2.4 encodingSize() `Rational smtrat::ExactlyOneCommanderEncoder::encodingSize ( const ConstraintT & constraint ) [virtual]`

Reimplemented from smtrat::PseudoBoolEncoder.

15.167.2.5 generateVarChain() `FormulaT smtrat::PseudoBoolEncoder::generateVarChain ( const std::set< carl::Variable > & vars, carl::FormulaType type ) [protected], [inherited]`

15.167.2.6 name() `std::string smtrat::ExactlyOneCommanderEncoder::name ( ) [inline], [virtual]`

Reimplemented from smtrat::PseudoBoolEncoder.

15.167.3 Field Documentation

15.167.3.1 problem_size `std::size_t smtrat::PseudoBoolEncoder::problem_size [inherited]`

15.168 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 ()

15.168.1 Constructor & Destructor Documentation

15.168.1.1 ExecutionState() `smtrat::execution::ExecutionState::ExecutionState ( ) [inline]`

15.168.2 Member Function Documentation

15.168.2.1 add_assertion() `void smtrat::execution::ExecutionState::add_assertion ( const FormulaT & formula ) [inline]`

15.168.2.2 add_objective() `void smtrat::execution::ExecutionState::add_objective ( const Poly & function, bool minimize ) [inline]`

15.168.2.3 add_soft_assertion()

```
void smtrat::execution::ExecutionState::add_soft_assertion (
    const FormulaT & formula,
    Rational weight,
    const std::string & id ) [inline]
```

15.168.2.4 annotate_name()

```
void smtrat::execution::ExecutionState::annotate_name (
    const std::string & name,
    const smtrat::FormulaT & f ) [inline]
```

15.168.2.5 assertions()

```
const auto& smtrat::execution::ExecutionState::assertions ( ) const
[inline]
```

15.168.2.6 enter_sat()

```
void smtrat::execution::ExecutionState::enter_sat (
    const smtrat::Model & model,
    const ObjectiveValues & objectiveValues ) [inline]
```

15.168.2.7 enter_unsat()

```
void smtrat::execution::ExecutionState::enter_unsat ( ) [inline]
```

15.168.2.8 has_annotated_name()

```
bool smtrat::execution::ExecutionState::has_annotated_name (
    const std::string & n ) [inline]
```

15.168.2.9 has_annotated_name_formula()

```
bool smtrat::execution::ExecutionState::has_annotated_↩
name_formula (
    const smtrat::FormulaT & f ) [inline]
```

15.168.2.10 has_assertion()

```
bool smtrat::execution::ExecutionState::has_assertion (
    const FormulaT & formula ) const [inline]
```

15.168.2.11 has_objective()

```
bool smtrat::execution::ExecutionState::has_objective (
    const Poly & function ) [inline]
```

15.168.2.12 has_soft_assertion()

```
bool smtrat::execution::ExecutionState::has_soft_assertion (
    const FormulaT & formula ) const [inline]
```

15.168.2.13 is_mode()

```
bool smtrat::execution::ExecutionState::is_mode (
    Mode mode ) const [inline]
```

15.168.2.14 logic()

`auto smtrat::execution::ExecutionState::logic ( ) const [inline]`

15.168.2.15 model()

`const smtrat::Model& smtrat::execution::ExecutionState::model ( ) const [inline]`

15.168.2.16 objectiveValues()

`const smtrat::ObjectiveValues& smtrat::execution::ExecutionState↩ ::objectiveValues ( ) const [inline]`

15.168.2.17 pop() [1/2]

`void smtrat::execution::ExecutionState::pop ( ) [inline]`

15.168.2.18 pop() [2/2]

```
void smtrat::execution::ExecutionState::pop (
    size_t n ) [inline]
```

15.168.2.19 push() [1/2]

`void smtrat::execution::ExecutionState::push ( ) [inline]`

15.168.2.20 push() [2/2]

```
void smtrat::execution::ExecutionState::push (
    size_t n ) [inline]
```

15.168.2.21 reset()

`void smtrat::execution::ExecutionState::reset ( ) [inline]`

15.168.2.22 reset_answer()

`void smtrat::execution::ExecutionState::reset_answer ( ) [inline]`

15.168.2.23 reset_assertions()

`void smtrat::execution::ExecutionState::reset_assertions ( ) [inline]`

15.168.2.24 set_add_annotated_name_handler()

```
void smtrat::execution::ExecutionState::set_add_↩
annotated_name_handler (
    std::function< void(const FormulaT &, const std::string &)> f ) [inline]
```

15.168.2.25 set_add_assertion_handler()

```
void smtrat::execution::ExecutionState::set_add_assertion↩
_handler (
    std::function< void(const FormulaT &)> f ) [inline]
```

15.168.2.26 set_add_objective_handler()

```
void smtrat::execution::ExecutionState::set_add_objective↩
_handler (
            std::function< void(const Poly &, bool)> f )  [inline]
```

15.168.2.27 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]
```

15.168.2.28 set_logic()

```
void smtrat::execution::ExecutionState::set_logic (
            carl::Logic logic )  [inline]
```

15.168.2.29 set_remove_annotated_name_handler()

```
void smtrat::execution::ExecutionState::set_↩
remove_annotated_name_handler (
            std::function< void(const FormulaT &)> f )  [inline]
```

15.168.2.30 set_remove_assertion_handler()

```
void smtrat::execution::ExecutionState::set_remove_↩
assertion_handler (
            std::function< void(const FormulaT &)> f )  [inline]
```

15.168.2.31 set_remove_objective_handler()

```
void smtrat::execution::ExecutionState::set_remove_↩
objective_handler (
            std::function< void(const Poly &)> f )  [inline]
```

15.168.2.32 set_remove_soft_assertion_handler()

```
void smtrat::execution::ExecutionState::set_↩
remove_soft_assertion_handler (
            std::function< void(const FormulaT &)> f )  [inline]
```

15.169 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< 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
- 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

- smtrat::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

15.169.1 Member Typedef Documentation

15.169.1.1 Value

```
typedef types::AttributeValue smtrat::parser::InstructionHandler::Value [inherited]
```

15.169.2 Constructor & Destructor Documentation

15.169.2.1 Executor()

```
template<typename Strategy >
smtrat::Executor< Strategy >::Executor (
            Strategy & solver ) [inline]
```

15.169.2.2 ~Executor()

```
template<typename Strategy >
smtrat::Executor< Strategy >::~Executor ( ) [inline]
```

15.169.3 Member Function Documentation

15.169.3.1 add()

```
template<typename Strategy >
void smtrat::Executor< Strategy >::add (
            const smtrat::FormulaT & f ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

15.169.3.2 addInstruction()

```
void smtrat::parser::InstructionHandler::addInstruction (
            std::function< void()> bind ) [inline], [inherited]
```

15.169.3.3 addObjective()

```
template<typename Strategy >
void smtrat::Executor< Strategy >::addObjective (
            const smtrat::Poly & p,
            smtrat::parser::OptimizationType ot ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

15.169.3.4 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.

15.169.3.5 annotateName()

```
template<typename Strategy >
void smtrat::Executor< Strategy >::annotateName (
            const smtrat::FormulaT & f,
            const std::string & name ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

15.169.3.6 check()

```
template<typename Strategy >
void smtrat::Executor< Strategy >::check ( ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

15.169.3.7 cleanModel()

```
void smtrat::parser::InstructionHandler::cleanModel (
    smtrat::Model & model ) const [inline], [inherited]
```

15.169.3.8 declareFun()

```
template<typename Strategy >
void smtrat::Executor< Strategy >::declareFun (
    const carl::Variable & ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

15.169.3.9 declareSort()

```
template<typename Strategy >
void smtrat::Executor< Strategy >::declareSort (
    const std::string & ,
    const unsigned & ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

15.169.3.10 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.

15.169.3.11 diagnostic() [1/2]

```
std::ostream& smtrat::parser::InstructionHandler::diagnostic ( )
[inline], [inherited]
```

15.169.3.12 diagnostic() [2/2]

```
void smtrat::parser::InstructionHandler::diagnostic (
    const std::string & s ) [inline], [inherited]
```

15.169.3.13 echo()

```
virtual void smtrat::parser::InstructionHandler::echo (
    const std::string & s ) [inline], [virtual], [inherited]
```

15.169.3.14 eliminateQuantifiers()

```
template<typename Strategy >
void smtrat::Executor< Strategy >::eliminateQuantifiers (
    const smtrat::qe::QEQuery & ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

15.169.3.15 error() `OutputWrapper smtrat::parser::InstructionHandler::error ( ) [inline], [inherited]`

15.169.3.16 exit() `template<typename Strategy >`
`void smtrat::Executor< Strategy >::exit ( ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

15.169.3.17 get_info() `const auto& smtrat::parser::InstructionHandler::get_info ( const std::string & key ) const [inline], [inherited]`

15.169.3.18 getAllModels() `template<typename Strategy >`
`void smtrat::Executor< Strategy >::getAllModels ( ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

15.169.3.19 getAssertions() `template<typename Strategy >`
`void smtrat::Executor< Strategy >::getAssertions ( ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

15.169.3.20 getAssignment() `template<typename Strategy >`
`void smtrat::Executor< Strategy >::getAssignment ( ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

15.169.3.21 getExitCode() `template<typename Strategy >`
`int smtrat::Executor< Strategy >::getExitCode ( ) const [inline]`

15.169.3.22 getInfo() `void smtrat::parser::InstructionHandler::getInfo ( const std::string & key ) [inline], [inherited]`

15.169.3.23 getModel() `template<typename Strategy >`
`void smtrat::Executor< Strategy >::getModel ( ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

15.169.3.24 getObjectives() `template<typename Strategy >`
`void smtrat::Executor< Strategy >::getObjectives ( ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

15.169.3.25 getOption()

```
void smtrat::parser::InstructionHandler::getOption (
    const std::string & key ) [inline], [inherited]
```

15.169.3.26 getProof()

```
template<typename Strategy >
void smtrat::Executor< Strategy >::getProof ( ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

15.169.3.27 getUnsatCore()

```
template<typename Strategy >
void smtrat::Executor< Strategy >::getUnsatCore ( ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

15.169.3.28 getValue()

```
template<typename Strategy >
void smtrat::Executor< Strategy >::getValue (
    const std::vector< carl::Variable > & ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

15.169.3.29 has_info()

```
bool smtrat::parser::InstructionHandler::has_info (
    const std::string & key ) const [inline], [inherited]
```

15.169.3.30 hasInstructions()

```
bool smtrat::parser::InstructionHandler::hasInstructions ( ) const
[inline], [inherited]
```

15.169.3.31 info()

```
OutputWrapper smtrat::parser::InstructionHandler::info ( ) [inline], [inherited]
```

15.169.3.32 option()

```
template<typename T >
T smtrat::parser::InstructionHandler::option (
    const std::string & key ) const [inline], [inherited]
```

15.169.3.33 pop()

```
template<typename Strategy >
void smtrat::Executor< Strategy >::pop (
    std::size_t n ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

15.169.3.34 printInstruction()

```
bool smtrat::parser::InstructionHandler::printInstruction ( )
const [inline], [inherited]
```

15.169.3.35 push()

```
template<typename Strategy >
void smtrat::Executor< Strategy >::push (
            std::size_t n )  [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

15.169.3.36 regular() [1/2]

```
std::ostream& smtrat::parser::InstructionHandler::regular ( )  [inline], [inherited]
```

15.169.3.37 regular() [2/2]

```
void smtrat::parser::InstructionHandler::regular (
            const std::string & s )  [inline], [inherited]
```

15.169.3.38 reset()

```
template<typename Strategy >
void smtrat::Executor< Strategy >::reset ( )  [inline], [virtual]
```

Reimplemented from smtrat::parser::InstructionHandler.

15.169.3.39 resetAssertions()

```
template<typename Strategy >
void smtrat::Executor< Strategy >::resetAssertions ( )  [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

15.169.3.40 runInstructions()

```
void smtrat::parser::InstructionHandler::runInstructions ( )  [inline], [inherited]
```

15.169.3.41 setArtificialVariables()

```
void smtrat::parser::InstructionHandler::setArtificial↩
Variables (
            std::vector< smtrat::ModelVariable > && vars )  [inline], [inherited]
```

15.169.3.42 setInfo()

```
void smtrat::parser::InstructionHandler::setInfo (
            const Attribute & attr )  [inline], [inherited]
```

15.169.3.43 setLogic()

```
template<typename Strategy >
void smtrat::Executor< Strategy >::setLogic (
            const carl::Logic & logic )  [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

15.169.3.44 setOption()

```
void smtrat::parser::InstructionHandler::setOption (
    const Attribute & option ) [inline], [inherited]
```

15.169.3.45 setStream()

```
void smtrat::parser::InstructionHandler::setStream (
    const std::string & s,
    std::ostream & os ) [inline], [protected], [inherited]
```

15.169.3.46 warn()

```
OutputWrapper smtrat::parser::InstructionHandler::warn ( ) [inline], [inherited]
```

15.169.4 Field Documentation

15.169.4.1 infos

```
VariantMap<std::string, Value> smtrat::parser::InstructionHandler::infos
[protected], [inherited]
```

15.169.4.2 lastAnswer

```
template<typename Strategy >
smtrat::Answer smtrat::Executor< Strategy >::lastAnswer
```

15.169.4.3 mDiagnostic

```
std::ostream smtrat::parser::InstructionHandler::mDiagnostic [protected],
[inherited]
```

15.169.4.4 mRegular

```
std::ostream smtrat::parser::InstructionHandler::mRegular [protected],
[inherited]
```

15.169.4.5 options

```
VariantMap<std::string, Value> smtrat::parser::InstructionHandler::options
[protected], [inherited]
```

15.169.4.6 streams

```
std::map<std::string, std::ofstream> smtrat::parser::InstructionHandler↩
::streams [protected], [inherited]
```

15.170 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

15.170.1 Member Function Documentation

15.170.1.1 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
```

15.171 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

15.171.1 Member Function Documentation

15.171.1.1 operator()()

```
std::optional< mcsat::Explanation > smtrat::mcsat::vs::Explanation↩
::operator() (
            const mcsat::Bookkeeping & data,
            carl::Variable var,
            const FormulasT & reason,
            bool  ) const
```

15.172 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

15.172.1 Member Function Documentation

15.172.1.1 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
```

15.173 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

15.173.1 Member Function Documentation

15.173.1.1 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
```

15.174 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 -> I', i.e.

15.174.1 Member Function Documentation

15.174.1.1 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 -> I', i.e.

'e1 & e2 ... en -> i', called "Explanation", in its clausal form '-e1 v -e2 ... en v 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

variableOrdering	Determine the order of variables, e.g. x1, x2, .. x10
data	Represent the assigned variables and their assigned values in different representations. These are a prefix of the `variableOrdering`, e.g. x1, x2, .. x5
var	The smallest/first variable in the `variableOrdering` that is not assigned by `data`, e.g. x6
reason	A collection of atoms that are inconsistent together or imply the `implication`. 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 `data`, e.g. x1∗x7+x6=0 (for x1 := 0) or x6∗(x7^2+1)>0 (equiv to x6>0, since x7^2+1 > 0)

15.175 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

15.175.1 Member Function Documentation

15.175.1.1 operator()()

```
std::optional< mcsat::Explanation > smtrat::mcsat::onecell::Explanation↩
::operator() (
            const mcsat::Bookkeeping & trail,
            carl::Variable var,
            const FormulasT & reason,
            bool ) const
```

15.176 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

15.176.1 Member Function Documentation

15.176.1.1 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
```

15.177 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

15.177.1 Constructor & Destructor Documentation

15.177.1.1 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]
```

15.177.2 Member Function Documentation

15.177.2.1 getExplanation()

```
template<class Settings >
std::optional<mcsat::Explanation> smtrat::mcsat::vs::ExplanationGenerator< Settings >::get↩
Explanation ( ) const  [inline]
```

15.178 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 > &explainAtomsinLvls) const

  Construct explanation in non-clausal form (without the impliedAtom/implication).

- mcsat::Explanation getExplanation (const FormulaT &impliedAtom) const

  Compute explanation in clausal form.

15.178.1 Constructor & Destructor Documentation

15.178.1.1 ExplanationGenerator()

```
smtrat::mcsat::nlsat::ExplanationGenerator::Explanation↩
Generator (
            const std::vector< FormulaT > & constraints,
            const std::vector< carl::Variable > & vars,
            carl::Variable ,
            const Model & model )  [inline]
```

15.178.2 Member Function Documentation

15.178.2.1 generateExplanation()

```
void smtrat::mcsat::nlsat::ExplanationGenerator::generate↩
Explanation (
            const FormulaT & impliedAtom,
            std::vector< FormulasT > & explainAtomsinLvls ) 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.

15.178.2.2 getExplanation()

```
mcsat::Explanation smtrat::mcsat::nlsat::ExplanationGenerator↩
::getExplanation (
            const FormulaT & impliedAtom ) const  [inline]
```

Compute explanation in clausal form.

15.179 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 isNary () 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

15.179.1 Constructor & Destructor Documentation

15.179.1.1 Expression() [1/8]

```
smtrat::expression::Expression::Expression (
    carl::Variable::Arg var ) [explicit]
```

15.179.1.2 Expression() [2/8]

```
smtrat::expression::Expression::Expression (
    ITEType _type,
    Expression && _if,
    Expression && _then,
    Expression && _else ) [explicit]
```

15.179.1.3 Expression() [3/8]

```
smtrat::expression::Expression::Expression (
    QuantifierType _type,
    std::vector< carl::Variable > && _variables,
    Expression && _expression ) [explicit]
```

15.179.1.4 Expression() [4/8]

```
smtrat::expression::Expression::Expression (
    UnaryType _type,
    Expression && _expression ) [explicit]
```

15.179.1.5 Expression() [5/8]

```
smtrat::expression::Expression::Expression (
    UnaryType _type,
    const Expression & _expression ) [inline], [explicit]
```

15.179.1.6 Expression() [6/8]

```
smtrat::expression::Expression::Expression (
    BinaryType _type,
    Expression && _lhs,
    Expression && _rhs ) [explicit]
```

15.179.1.7 Expression() [7/8]

```
smtrat::expression::Expression::Expression (
    NaryType _type,
    Expressions && _expressions ) [explicit]
```

15.179.1.8 Expression() [8/8]

```
smtrat::expression::Expression::Expression (
    NaryType _type,
    const std::initializer_list< Expression > & _expressions ) [explicit]
```

15.179.2 Member Function Documentation

15.179.2.1 getBinary()

`const BinaryExpression& smtrat::expression::Expression::getBinary ( ) const`

15.179.2.2 getContent()

`const ExpressionContent& smtrat::expression::Expression::getContent ( ) const [inline]`

15.179.2.3 getContentPtr()

`const ExpressionContent* smtrat::expression::Expression::getContent↩Ptr ( ) const [inline]`

15.179.2.4 getITE()

`const ITEExpression& smtrat::expression::Expression::getITE ( ) const`

15.179.2.5 getNary()

`const NaryExpression& smtrat::expression::Expression::getNary ( ) const`

15.179.2.6 getNegationPtr()

`const ExpressionContent* smtrat::expression::Expression::get↩NegationPtr ( ) const`

15.179.2.7 getQuantifier()

`const QuantifierExpression& smtrat::expression::Expression::get↩Quantifier ( ) const`

15.179.2.8 getUnary()

`const UnaryExpression& smtrat::expression::Expression::getUnary ( ) const`

15.179.2.9 isBinary()

`bool smtrat::expression::Expression::isBinary ( ) const`

15.179.2.10 isITE()

`bool smtrat::expression::Expression::isITE ( ) const`

15.179.2.11 isNary()

`bool smtrat::expression::Expression::isNary ( ) const`

15.179.2.12 isQuantifier()

`bool smtrat::expression::Expression::isQuantifier ( ) const`

15.179.2.13 isUnary()

`bool smtrat::expression::Expression::isUnary ( ) const`

15.179.2.14 operator"!=() `bool smtrat::expression::Expression::operator!= ( const Expression & expr ) const`

15.179.2.15 operator<() `bool smtrat::expression::Expression::operator< ( const Expression & expr ) const`

15.179.2.16 operator==() `bool smtrat::expression::Expression::operator== ( const Expression & expr ) const`

15.179.3 Friends And Related Function Documentation

15.179.3.1 ExpressionModifier `friend class ExpressionModifier [friend]`

15.179.3.2 ExpressionPool `friend class ExpressionPool [friend]`

15.179.3.3 std::hash< smtrat::expression::Expression > `friend struct std::hash< smtrat::expression::Expression > [friend]`

15.180 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 >

15.180.1 Member Typedef Documentation

15.180.1.1 Content `typedef boost::variant< carl::Variable, ITEExpression, QuantifierExpression, UnaryExpression, BinaryExpression, NaryExpression > smtrat::expression::ExpressionContent::Content`

15.180.2 Constructor & Destructor Documentation

15.180.2.1 ExpressionContent() `template<typename... T> smtrat::expression::ExpressionContent::ExpressionContent ( T &&... t ) [inline]`

15.180.3 Friends And Related Function Documentation

15.180.3.1 std::hash< Content > `friend struct std::hash< Content > [friend]`

15.180.4 Field Documentation

15.180.4.1 content `Content smtrat::expression::ExpressionContent::content`

15.180.4.2 hash `std::size_t smtrat::expression::ExpressionContent::hash`

15.180.4.3 id `std::size_t smtrat::expression::ExpressionContent::id`

15.180.4.4 negation `const ExpressionContent* smtrat::expression::ExpressionContent::negation`

15.181 smtrat::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

15.181.1 Member Function Documentation

15.181.1.1 convert() `FormulaT smtrat::expression::ExpressionConverter::convert ( const Expression & expr ) [inline], [protected]`

15.181.1.2 operator()() [1/7] `FormulaT smtrat::expression::ExpressionConverter::operator() ( carl::Variable::Arg var ) [inline]`

15.181.1.3 operator()() [2/7] `FormulaT smtrat::expression::ExpressionConverter::operator() ( const BinaryExpression & expr ) [inline]`

15.181.1.4 operator()() [3/7] `FormulaT smtrat::expression::ExpressionConverter::operator() ( const Expression & expr ) [inline]`

15.181.1.5 operator()() [4/7]

```
FormulaT smtrat::expression::ExpressionConverter::operator() (
    const ITEExpression & expr ) [inline]
```

15.181.1.6 operator()() [5/7]

```
FormulaT smtrat::expression::ExpressionConverter::operator() (
    const NaryExpression & expr ) [inline]
```

15.181.1.7 operator()() [6/7]

```
FormulaT smtrat::expression::ExpressionConverter::operator() (
    const QuantifierExpression & expr ) [inline]
```

15.181.1.8 operator()() [7/7]

```
FormulaT smtrat::expression::ExpressionConverter::operator() (
    const UnaryExpression & expr ) [inline]
```

15.181.2 Field Documentation

15.181.2.1 mGlobalFormulas

```
FormulasT smtrat::expression::ExpressionConverter::mGlobalFormulas
[protected]
```

15.182 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)

15.182.1 Member Typedef Documentation

15.182.1.1 VisitorFunction

```
typedef std::function<const ExpressionContent*(const Expression&)>
smtrat::expression::ExpressionModifier::VisitorFunction
```

15.182.2 Member Function Documentation

15.182.2.1 operator()() [1/6]

```
const ExpressionContent* smtrat::expression::ExpressionModifier↩
::operator() (
    carl::Variable::Arg ) [inline]
```

15.182.2.2 operator()() [2/6]

```
const ExpressionContent* smtrat::expression::ExpressionModifier↩
::operator() (
    const BinaryExpression & expr ) [inline]
```

15.182.2.3 operator()() [3/6]

```
const ExpressionContent* smtrat::expression::ExpressionModifier↩
::operator() (
            const ITEExpression & expr ) [inline]
```

15.182.2.4 operator()() [4/6]

```
const ExpressionContent* smtrat::expression::ExpressionModifier↩
::operator() (
            const NaryExpression & expr ) [inline]
```

15.182.2.5 operator()() [5/6]

```
const ExpressionContent* smtrat::expression::ExpressionModifier↩
::operator() (
            const QuantifierExpression & expr ) [inline]
```

15.182.2.6 operator()() [6/6]

```
const ExpressionContent* smtrat::expression::ExpressionModifier↩
::operator() (
            const UnaryExpression & expr ) [inline]
```

15.182.2.7 setPost()

```
void smtrat::expression::ExpressionModifier::setPost (
            const VisitorFunction & f ) [inline]
```

15.182.2.8 setPre()

```
void smtrat::expression::ExpressionModifier::setPre (
            const VisitorFunction & f ) [inline]
```

15.182.2.9 visit()

```
Expression smtrat::expression::ExpressionModifier::visit (
            const Expression & expression ) [inline]
```

15.183 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

15.183.1 Constructor & Destructor Documentation

15.183.1.1 ExpressionPool() `smtrat::expression::ExpressionPool::ExpressionPool ( ) [inline], [protected]`

15.183.2 Member Function Documentation

15.183.2.1 add()

```
const ExpressionContent* smtrat::expression::ExpressionPool::add (
    ExpressionContent * _ec ) [protected]
```

15.183.2.2 create() [1/8]

```
template<typename... Args>
static const ExpressionContent* smtrat::expression::ExpressionPool::create (
    Args &&... args ) [inline], [static]
```

15.183.2.3 create() [2/8]

```
const ExpressionContent* smtrat::expression::ExpressionPool::create (
    BinaryType _type,
    Expression && _lhs,
    Expression && _rhs )
```

15.183.2.4 create() [3/8]

```
const ExpressionContent* smtrat::expression::ExpressionPool::create (
    carl::Variable::Arg var )
```

15.183.2.5 create() [4/8]

```
const ExpressionContent* smtrat::expression::ExpressionPool::create (
    ITEType _type,
    Expression && _if,
    Expression && _then,
    Expression && _else )
```

15.183.2.6 create() [5/8]

```
const ExpressionContent* smtrat::expression::ExpressionPool::create (
    NaryType _type,
    const std::initializer_list< Expression > & _expressions )
```

15.183.2.7 create() [6/8]

```
const ExpressionContent* smtrat::expression::ExpressionPool::create (
    NaryType _type,
    Expressions && _expressions )
```

15.183.2.8 create() [7/8]

```
const ExpressionContent* smtrat::expression::ExpressionPool::create (
    QuantifierType _type,
    std::vector< carl::Variable > && _variables,
    Expression && _expression )
```

15.183.2.9 create() [8/8]

```
const ExpressionContent* smtrat::expression::ExpressionPool::create (
    UnaryType _type,
    Expression && _expression )
```

15.183.3 Friends And Related Function Documentation

15.183.3.1 ExpressionModifier

```
friend class ExpressionModifier [friend]
```

15.184 smtrat::parser::ParserState::ExpressionScope Struct Reference

```
#include <ParserState.h>
```

Public Member Functions

- ExpressionScope (const ParserState &state)
- void discharge (ParserState &state)

15.184.1 Constructor & Destructor Documentation

15.184.1.1 ExpressionScope()

```
smtrat::parser::ParserState::ExpressionScope::ExpressionScope (
    const ParserState & state ) [inline]
```

15.184.2 Member Function Documentation

15.184.2.1 discharge()

```
void smtrat::parser::ParserState::ExpressionScope::discharge (
    ParserState & state ) [inline]
```

15.185 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

15.185.1 Member Function Documentation

15.185.1.1 operator()() [1/2]

```
template<typename T >
bool smtrat::expression::ExpressionTypeChecker< T >::operator() (
    const T & ) const [inline]
```

15.185.1.2 operator()() [2/2]

```
template<typename T >
template<typename T2 >
bool smtrat::expression::ExpressionTypeChecker< T >::operator() (
    const T2 & ) const [inline]
```

15.186 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)

15.186.1 Member Typedef Documentation

15.186.1.1 VisitorFunction

```
typedef std::function<void(const Expression&)> smtrat::expression::ExpressionVisitc
```

15.186.2 Member Function Documentation

15.186.2.1 operator()() [1/6]

```
void smtrat::expression::ExpressionVisitor::operator() (
    carl::Variable::Arg ) [inline]
```

15.186.2.2 operator()() [2/6]

```
void smtrat::expression::ExpressionVisitor::operator() (
    const BinaryExpression & expr ) [inline]
```

15.186.2.3 operator()() [3/6]

```
void smtrat::expression::ExpressionVisitor::operator() (
    const ITEExpression & expr ) [inline]
```

15.186.2.4 operator()() [4/6]

```
void smtrat::expression::ExpressionVisitor::operator() (
    const NaryExpression & expr ) [inline]
```

15.186.2.5 operator()() [5/6]

```
void smtrat::expression::ExpressionVisitor::operator() (
    const QuantifierExpression & expr ) [inline]
```

15.186.2.6 operator()() [6/6]

```
void smtrat::expression::ExpressionVisitor::operator() (
    const UnaryExpression & expr ) [inline]
```

15.186.2.7 setPost()

```
void smtrat::expression::ExpressionVisitor::setPost (
    const VisitorFunction & f ) [inline]
```

15.186.2.8 setPre()

```
void smtrat::expression::ExpressionVisitor::setPre (
    const VisitorFunction & f ) [inline]
```

15.186.2.9 visit()

```
void smtrat::expression::ExpressionVisitor::visit (
    const Expression & expression ) [inline]
```

15.187 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>
```

15.187.1 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.

15.188 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

15.188.1 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.

15.188.2 Member Typedef Documentation

15.188.2.1 Extractor

```
template<typename Objects>
using smtrat::mcsat::variableordering::detail::FeatureCollector< Objects >::Extractor = std↩
::function<double(Objects, carl::Variable)>
```

15.188.2.2 Valuation template<typename Objects>

```
using smtrat::mcsat::variableordering::detail::FeatureCollector< Objects >::Valuation = std↩
::vector<double>
```

15.188.3 Member Function Documentation

15.188.3.1 addFeature() template<typename Objects>

```
void smtrat::mcsat::variableordering::detail::FeatureCollector< Objects >::addFeature (
            Extractor && e,
            std::size_t level,
            double weight )  [inline]
```

15.188.3.2 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]
```

15.188.3.3 valuateVariable() template<typename Objects>

```
Valuation smtrat::mcsat::variableordering::detail::FeatureCollector< Objects >::valuate↩
Variable (
            const Objects & o,
            carl::Variable v ) const  [inline]
```

15.188.4 Field Documentation

15.188.4.1 mFeatures template<typename Objects>

```
std::vector<std::tuple<Extractor,std::size_t,double> > smtrat::mcsat::variableordering::detail::FeatureCollec
Objects >::mFeatures
```

15.189 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

15.189.1 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.

15.189.2 Constructor & Destructor Documentation

15.189.2.1 fixedsize_allocator() [1/3]

```
template<typename T >
smtrat::fixedsize_allocator< T >::fixedsize_allocator ( ) [inline]
```

15.189.2.2 fixedsize_allocator() [2/3]

```
template<typename T >
smtrat::fixedsize_allocator< T >::fixedsize_allocator (
            const fixedsize_allocator< T > & ) [default]
```

15.189.2.3 fixedsize_allocator() [3/3]

```
template<typename T >
template<typename U >
smtrat::fixedsize_allocator< T >::fixedsize_allocator (
            const fixedsize_allocator< U > & ) [inline]
```

15.189.3 Member Function Documentation

15.189.3.1 operator"!=()

```
template<typename T >
template<typename T2 >
CONSTEXPR bool smtrat::fixedsize_allocator< T >::operator!= (
            const fixedsize_allocator< T2 > & ) const [inline]
```

15.189.3.2 operator==()

```
template<typename T >
template<typename T2 >
CONSTEXPR bool smtrat::fixedsize_allocator< T >::operator== (
            const fixedsize_allocator< T2 > & ) const [inline]
```

15.190 smtrat::fixedsize_allocator< void > Class Template 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.

15.190.1 Detailed Description

template<>
class smtrat::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.

15.190.2 Field Documentation

15.190.2.1 elements `T std::allocator< T >::elements [inherited]`

STL member.

15.191 smtrat::impl::fixedsize_allocator_freelists Class Reference

`#include <alloc.h>`

Static Public Attributes

- static fixedsize_allocator_freelists INSTANCE

15.191.1 Field Documentation

15.191.1.1 INSTANCE `fixedsize_allocator_freelists smtrat::impl::fixedsize_allocator_freelists↩ ::INSTANCE [static]`

15.192 smtrat::impl::fixedsize_allocator_impl< T, SizeShift, LargeEnough, SmallEnough > Class Template Reference

`#include <alloc.h>`

15.193 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 T & 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)

Data Fields

- const typedef T * const_pointer
- const typedef T & const_reference

15.193.1 Member Typedef Documentation

15.193.1.1 difference_type

```
template<typename T , std::size_t SizeShift>
typedef std::ptrdiff_t smtrat::impl::fixedsize_allocator_impl< T, SizeShift, true, true >↩
::difference_type
```

15.193.1.2 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
```

15.193.1.3 pointer

```
template<typename T , std::size_t SizeShift>
typedef T* smtrat::impl::fixedsize_allocator_impl< T, SizeShift, true, true >::pointer
```

15.193.1.4 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
```

15.193.1.5 reference

```
template<typename T , std::size_t SizeShift>
typedef T& smtrat::impl::fixedsize_allocator_impl< T, SizeShift, true, true >::reference
```

15.193.1.6 size_type

```
template<typename T , std::size_t SizeShift>
typedef std::size_t smtrat::impl::fixedsize_allocator_impl< T, SizeShift, true, true >::size_type
```

15.193.1.7 value_type

```
template<typename T , std::size_t SizeShift>
typedef T smtrat::impl::fixedsize_allocator_impl< T, SizeShift, true, true >::value_type
```

15.193.2 Member Function Documentation

15.193.2.1 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]
```

15.193.2.2 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]
```

15.193.2.3 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]
```

15.193.2.4 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]
```

15.193.2.5 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]
```

15.193.2.6 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]
```

15.193.2.7 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]
```

15.193.3 Field Documentation

15.193.3.1 const_pointer

```
template<typename T , std::size_t SizeShift>
const typedef T* smtrat::impl::fixedsize_allocator_impl< T, SizeShift, true, true >::const_↩
pointer
```

15.193.3.2 const_reference

```
template<typename T , std::size_t SizeShift>
const typedef T& smtrat::impl::fixedsize_allocator_impl< T, SizeShift, true, true >::const_↩
reference
```

15.194 smtrat::impl::fixedsize_allocator_size_helper< T, Size > Class Template Reference

```
#include <alloc.h>
```

15.195 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.

15.195.1 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.

15.195.2 Constructor & Destructor Documentation

15.195.2.1 fixedsize_freelist()

```
template<std::size_t ChunkSizeShift>
smtrat::fixedsize_freelist< ChunkSizeShift >::fixedsize_freelist ( ) [inline], [explicit]
```

15.195.2.2 ~fixedsize_freelist()

```
template<std::size_t ChunkSizeShift>
smtrat::fixedsize_freelist< ChunkSizeShift >::~fixedsize_freelist ( ) [inline]
```

15.195.3 Member Function Documentation

15.195.3.1 alloc()

```
template<std::size_t ChunkSizeShift>
void* smtrat::fixedsize_freelist< ChunkSizeShift >::alloc ( ) [inline]
```

Allocate an object of size 1<<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.

15.195.3.2 free()

```
template<std::size_t ChunkSizeShift>
void smtrat::fixedsize_freelist< ChunkSizeShift >::free (
            void * ptr ) [inline]
```

Freeing means just prepending the free'd element to the front of the freelist.

15.196 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>
```

15.196.1 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.

15.197 smtrat::impl::fixedsize_freelists< SizeCurrent, SizeMax, true > Class Template Reference

```
#include <alloc.h>
```

15.198 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

15.198.1 Detailed Description

template<typename T>
struct smtrat::parser::FixedWidthConstant< T >

Represents a constant of a fixed width.

15.198.2 Constructor & Destructor Documentation

15.198.2.1 FixedWidthConstant() [1/2] template<typename T>

```
smtrat::parser::FixedWidthConstant< T >::FixedWidthConstant ( ) [inline]
```

15.198.2.2 FixedWidthConstant() [2/2] template<typename T>

```
smtrat::parser::FixedWidthConstant< T >::FixedWidthConstant (
            const T & value,
            std::size_t width ) [inline]
```

15.198.3 Member Function Documentation

15.198.3.1 operator<() template<typename T>

```
bool smtrat::parser::FixedWidthConstant< T >::operator< (
            const FixedWidthConstant< T > & fwc ) const [inline]
```

15.198.4 Field Documentation

15.198.4.1 value template<typename T>

```
T smtrat::parser::FixedWidthConstant< T >::value
```

15.198.4.2 width template<typename T>

```
std::size_t smtrat::parser::FixedWidthConstant< T >::width
```

15.199 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 weight, const std::string &name)
- 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 &p, smtrat::parser::OptimizationType ot)
- 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< 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
- 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

15.199.1 Member Typedef Documentation

15.199.1.1 Value

```
typedef types::AttributeValue smtrat::parser::InstructionHandler::Value [inherited]
```

15.199.2 Member Function Documentation

15.199.2.1 add()

```
void smtrat::parseformula::FormulaCollector::add (
    const smtrat::FormulaT & f ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

15.199.2.2 addInstruction()

```
void smtrat::parser::InstructionHandler::addInstruction (
    std::function< void() > bind ) [inline], [inherited]
```

15.199.2.3 addObjective()

```
void smtrat::parseformula::FormulaCollector::addObjective (
    const smtrat::Poly & p,
    smtrat::parser::OptimizationType ot ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

15.199.2.4 addSoft()

```
void smtrat::parseformula::FormulaCollector::addSoft (
    const FormulaT & f,
    Rational weight,
    const std::string & name ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

15.199.2.5 annotateName()

```
void smtrat::parseformula::FormulaCollector::annotateName (
    const smtrat::FormulaT & ,
    const std::string & ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

15.199.2.6 check()

```
void smtrat::parseformula::FormulaCollector::check ( ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

15.199.2.7 cleanModel()

```
void smtrat::parser::InstructionHandler::cleanModel (
    smtrat::Model & model ) const [inline], [inherited]
```

15.199.2.8 declareFun()

```
void smtrat::parseformula::FormulaCollector::declareFun (
    const carl::Variable & ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

15.199.2.9 declareSort()

```
void smtrat::parseformula::FormulaCollector::declareSort (
    const std::string & ,
    const unsigned & ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

15.199.2.10 defineSort()

```
void smtrat::parseformula::FormulaCollector::defineSort (
    const std::string & ,
    const std::vector< std::string > & ,
    const carl::Sort & ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

15.199.2.11 diagnostic() [1/2]

```
std::ostream& smtrat::parser::InstructionHandler::diagnostic ( )
[inline], [inherited]
```

15.199.2.12 diagnostic() [2/2]

```
void smtrat::parser::InstructionHandler::diagnostic (
    const std::string & s ) [inline], [inherited]
```

15.199.2.13 echo()

```
virtual void smtrat::parser::InstructionHandler::echo (
    const std::string & s ) [inline], [virtual], [inherited]
```

15.199.2.14 eliminateQuantifiers()

```
void smtrat::parseformula::FormulaCollector::eliminateQuantifiers
(
    const smtrat::qe::QEQuery & ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

15.199.2.15 error()

```
OutputWrapper smtrat::parser::InstructionHandler::error ( ) [inline],
[inherited]
```

15.199.2.16 exit() `void smtrat::parseformula::FormulaCollector::exit ( ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

15.199.2.17 get_info() `const auto& smtrat::parser::InstructionHandler::get_info ( const std::string & key ) const [inline], [inherited]`

15.199.2.18 getAllModels() `void smtrat::parseformula::FormulaCollector::getAllModels ( ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

15.199.2.19 getAssertions() `void smtrat::parseformula::FormulaCollector::getAssertions ( ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

15.199.2.20 getAssignment() `void smtrat::parseformula::FormulaCollector::getAssignment ( ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

15.199.2.21 getExitCode() `int smtrat::parseformula::FormulaCollector::getExitCode ( ) const [inline]`

15.199.2.22 getFormula() `FormulaT smtrat::parseformula::FormulaCollector::getFormula ( ) const [inline]`

15.199.2.23 getInfo() `void smtrat::parser::InstructionHandler::getInfo ( const std::string & key ) [inline], [inherited]`

15.199.2.24 getLogic() `auto smtrat::parseformula::FormulaCollector::getLogic ( ) const [inline]`

15.199.2.25 getModel() `void smtrat::parseformula::FormulaCollector::getModel ( ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

15.199.2.26 getObjectives()

`void smtrat::parseformula::FormulaCollector::getObjectives ( ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

15.199.2.27 getOption()

`void smtrat::parser::InstructionHandler::getOption ( const std::string & key ) [inline], [inherited]`

15.199.2.28 getProof()

`void smtrat::parseformula::FormulaCollector::getProof ( ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

15.199.2.29 getUnsatCore()

`void smtrat::parseformula::FormulaCollector::getUnsatCore ( ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

15.199.2.30 getValue()

`void smtrat::parseformula::FormulaCollector::getValue ( const std::vector< carl::Variable > & ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

15.199.2.31 has_info()

`bool smtrat::parser::InstructionHandler::has_info ( const std::string & key ) const [inline], [inherited]`

15.199.2.32 hasInstructions()

`bool smtrat::parser::InstructionHandler::hasInstructions ( ) const [inline], [inherited]`

15.199.2.33 info()

`OutputWrapper smtrat::parser::InstructionHandler::info ( ) [inline], [inherited]`

15.199.2.34 option()

`template<typename T > T smtrat::parser::InstructionHandler::option ( const std::string & key ) const [inline], [inherited]`

15.199.2.35 pop()

`void smtrat::parseformula::FormulaCollector::pop ( std::size_t ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

15.199.2.36 printInstruction()

`bool smtrat::parser::InstructionHandler::printInstruction ( ) const [inline], [inherited]`

15.199.2.37 push()

`void smtrat::parseformula::FormulaCollector::push ( std::size_t ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

15.199.2.38 regular() [1/2]

`std::ostream& smtrat::parser::InstructionHandler::regular ( ) [inline], [inherited]`

15.199.2.39 regular() [2/2]

`void smtrat::parser::InstructionHandler::regular ( const std::string & s ) [inline], [inherited]`

15.199.2.40 reset()

`void smtrat::parseformula::FormulaCollector::reset ( ) [inline], [virtual]`

Reimplemented from smtrat::parser::InstructionHandler.

15.199.2.41 resetAssertions()

`void smtrat::parseformula::FormulaCollector::resetAssertions ( ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

15.199.2.42 runInstructions()

`void smtrat::parser::InstructionHandler::runInstructions ( ) [inline], [inherited]`

15.199.2.43 setArtificialVariables()

`void smtrat::parser::InstructionHandler::setArtificialVariables ( std::vector< smtrat::ModelVariable > && vars ) [inline], [inherited]`

15.199.2.44 setInfo()

`void smtrat::parser::InstructionHandler::setInfo ( const Attribute & attr ) [inline], [inherited]`

15.199.2.45 setLogic()

`void smtrat::parseformula::FormulaCollector::setLogic ( const carl::Logic & l ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

15.199.2.46 setOption()

```
void smtrat::parser::InstructionHandler::setOption (
    const Attribute & option ) [inline], [inherited]
```

15.199.2.47 setStream()

```
void smtrat::parser::InstructionHandler::setStream (
    const std::string & s,
    std::ostream & os ) [inline], [protected], [inherited]
```

15.199.2.48 warn()

```
OutputWrapper smtrat::parser::InstructionHandler::warn ( ) [inline], [inherited]
```

15.199.3 Field Documentation

15.199.3.1 infos

```
VariantMap<std::string, Value> smtrat::parser::InstructionHandler::infos
[protected], [inherited]
```

15.199.3.2 mDiagnostic

```
std::ostream smtrat::parser::InstructionHandler::mDiagnostic [protected],
[inherited]
```

15.199.3.3 mRegular

```
std::ostream smtrat::parser::InstructionHandler::mRegular [protected],
[inherited]
```

15.199.3.4 options

```
VariantMap<std::string, Value> smtrat::parser::InstructionHandler::options
[protected], [inherited]
```

15.199.3.5 streams

```
std::map<std::string, std::ofstream> smtrat::parser::InstructionHandler↩
::streams [protected], [inherited]
```

15.200 smtrat::ICPModule< Settings >::formulaPtrComp Struct Reference

```
#include <ICPModule.h>
```

Public Member Functions

- bool operator() (const FormulaT &_lhs, const FormulaT &_rhs) const

15.200.1 Member Function Documentation

15.200.1.1 operator()()

```
template<class Settings>
bool smtrat::ICPModule< Settings >::formulaPtrComp::operator() (
    const FormulaT & _lhs,
    const FormulaT & _rhs ) const [inline]
```

15.201 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)

15.201.1 Detailed Description

Stores a formula along with its origins.

15.201.2 Constructor & Destructor Documentation

15.201.2.1 FormulaWithOrigins() [1/4]

```
smtrat::FormulaWithOrigins::FormulaWithOrigins ( )
```

15.201.2.2 FormulaWithOrigins() [2/4]

```
smtrat::FormulaWithOrigins::FormulaWithOrigins (
    const FormulaT & _formula ) [inline]
```

Constructs a formula with empty origins.

Parameters

_formula	The formula of the formula with origins to construct.

15.201.2.3 FormulaWithOrigins() [3/4]

```
smtrat::FormulaWithOrigins::FormulaWithOrigins (
    const FormulaT & _formula,
    const std::shared_ptr< FormulasT > & _origins ) [inline]
```

Constructs a formula with the given origins.

Parameters

_formula	The formula of the formula with origins to construct.
_origins	The origins of the formula with origins to construct.

15.201.2.4 FormulaWithOrigins() [4/4]

```
smtrat::FormulaWithOrigins::FormulaWithOrigins (
    const FormulaWithOrigins & )
```

15.201.3 Member Function Documentation

15.201.3.1 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.

15.201.3.2 formula() `const FormulaT& smtrat::FormulaWithOrigins::formula ( ) const [inline]`

Returns A constant reference to the formula.

15.201.3.3 hasOrigins() `bool smtrat::FormulaWithOrigins::hasOrigins ( ) const [inline]`

Returns true, if this sub-formula of the module input has origins.

15.201.3.4 origins() `const FormulasT& smtrat::FormulaWithOrigins::origins ( ) const [inline]`

Returns A constant reference to the origins.

15.201.3.5 setDeducted() `void smtrat::FormulaWithOrigins::setDeducted ( bool _deducted ) const [inline]`

Sets the deduction flag to the given value.

Parameters

_deducted	The value to set the deduction flag to.

15.201.4 Friends And Related Function Documentation

15.201.4.1 ModuleInput `friend class ModuleInput [friend]`

15.201.4.2 operator< `bool operator< ( const FormulaWithOrigins & _fwoA, const FormulaWithOrigins & _fwoB ) [friend]`

Parameters

_fwoA	The first formula with origins to compare.
_fwoB	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.

15.201.4.3 operator==

```
bool operator== (
    const FormulaWithOrigins & _fwoA,
    const FormulaWithOrigins & _fwoB ) [friend]
```

Parameters

_fwoA	The first formula with origins to compare.
_fwoB	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.

15.202 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.

15.202.1 Detailed Description

template<typename VertexProperty, typename EdgeProperty, bool UniqueVertices = true>
class smtrat::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.

15.202.2 Member Typedef Documentation

15.202.2.1 Vertex

```
template<typename VertexProperty, typename EdgeProperty, bool UniqueVertices
= true>
using smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Vertex =
std::size_t
```

Internal type of a vertex.

15.202.3 Member Function Documentation

15.202.3.1 begin()

```
template<typename VertexProperty, typename EdgeProperty, bool Unique↩
Vertices = true>
Vertex smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::begin ( )
const [inline]
```

Returns the first vertex.

15.202.3.2 end()

```
template<typename VertexProperty, typename EdgeProperty, bool UniqueVertices
= true>
Vertex smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::end ( )
const [inline]
```

Returns the past-the-last vertex.

15.202.3.3 newEdge() [1/2]

```
template<typename VertexProperty, typename EdgeProperty, bool
UniqueVertices = true>
Edge& smtrat::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.

15.202.3.4 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.

15.202.3.5 newVertex()

```
template<typename VertexProperty, typename EdgeProperty, bool Unique↩
Vertices = 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.

15.202.3.6 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.

15.202.3.7 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.

15.202.3.8 out() [1/2]

```
template<typename VertexProperty, typename EdgeProperty, bool Unique↩
Vertices = true>
const std::vector<Edge>& smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, Unique↩
Vertices >::out (
            const VertexProperty & v ) const  [inline]
```

Retrieves all outgoing edges for the given vertex.

15.202.3.9 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.

15.202.3.10 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.

15.202.3.11 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.

15.202.4 Friends And Related Function Documentation

15.202.4.1 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.

15.203 smtrat::FouMoModule< Settings > Class Template Reference

A module which applies the Fourier-Motzkin algorithm.

```
#include <FouMoModule.h>
```

Public Types

- typedef Settings SettingsType
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- FouMoModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=NULL)
- ~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)

  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 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 _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)

  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 &_branching↩Value) 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 _preferLeft↩Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩::vector< FormulaT >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩CaseWeak=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.

15.203.1 Detailed Description

template<typename Settings>
class smtrat::FouMoModule< Settings >

A module which applies the Fourier-Motzkin algorithm.

15.203.2 Member Typedef Documentation

15.203.2.1 SettingsType

```
template<typename Settings >
typedef Settings smtrat::FouMoModule< Settings >::SettingsType
```

15.203.3 Member Enumeration Documentation

15.203.3.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

15.203.4 Constructor & Destructor Documentation

15.203.4.1 FouMoModule()

```
template<class Settings >
smtrat::FouMoModule< Settings >::FouMoModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = NULL )
```

15.203.4.2 ~FouMoModule()

```
template<typename Settings >
smtrat::FouMoModule< Settings >::~FouMoModule ( ) [inline]
```

15.203.5 Member Function Documentation

15.203.5.1 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.

15.203.5.2 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

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

15.203.5.3 addCore()

```
template<class 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

_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.

15.203.5.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
            const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.203.5.5 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.

15.203.5.6 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.

15.203.5.7 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.

15.203.5.8 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.

15.203.5.9 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

_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.

15.203.5.10 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

_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.

15.203.5.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const  [inline],
[inherited]
```

Returns All satisfying assignments, if existent.

15.203.5.12 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.

15.203.5.13 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.

15.203.5.14 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.

15.203.5.15 branchAt() [1/6]

```
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]
```

15.203.5.16 branchAt() [2/6]

```
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]
```

15.203.5.17 branchAt() [3/6]

```
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]
```

15.203.5.18 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.203.5.19 branchAt() [5/6] `template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>`

```
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            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]
```

15.203.5.20 branchAt() [6/6] `template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>`

```
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

15.203.5.21 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in smtrat::PModule.

15.203.5.22 checkCore()

```
template<class Settings >
Answer smtrat::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 smtrat::Module.

15.203.5.23 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
    std::vector< FormulaSetT >::const_iterator _infsubset,
    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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.203.5.24 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.

15.203.5.25 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.203.5.26 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.203.5.27 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.203.5.28 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.203.5.29 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.203.5.30 collectOrigins() [1/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

15.203.5.31 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

_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.

15.203.5.32 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.203.5.33 constraintsToInform()

```
const carl::FastSet<FormulaT>& smtrat::Module::constraints↩
ToInform ( ) const [inline], [inherited]
```

Returns The constraints which the backends must be informed about.

15.203.5.34 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::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.203.5.35 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)

15.203.5.36 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

_constraint	The constraint to remove from internal data structures.

15.203.5.37 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.203.5.38 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.203.5.39 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.203.5.40 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceived↩
VariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.203.5.41 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBest↩
Origin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.203.5.42 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.

15.203.5.43 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.

15.203.5.44 freeSplittingVariable() `static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

15.203.5.45 generateTrivialInfeasibleSubset() `void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

15.203.5.46 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

15.203.5.47 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

15.203.5.48 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

15.203.5.49 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.203.5.50 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.203.5.51 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.

15.203.5.52 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.203.5.53 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.203.5.54 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.203.5.55 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 reveived formula, if the Boolean is true.

Reimplemented in smtrat::PModule.

15.203.5.56 hasLemmas()

`bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

15.203.5.57 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).

15.203.5.58 id()

`std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns A unique ID to identify this module instance.

15.203.5.59 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.

15.203.5.60 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.203.5.61 informBackends()

```
void smtrat::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.

15.203.5.62 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, and smtrat::ICPModule< ICPSettings1 >.

15.203.5.63 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.203.5.64 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::FPPModule< Settings >.

15.203.5.65 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.203.5.66 isLemmaLevel() `bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.203.5.67 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().

15.203.5.68 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.

15.203.5.69 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.203.5.70 model()

`const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.203.5.71 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

_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.

15.203.5.72 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.

15.203.5.73 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

15.203.5.74 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
    const FormulaT & _origin ) const [protected], [inherited]
```

15.203.5.75 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula. TODO: disable this method

15.203.5.76 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula.

15.203.5.77 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to the backends of this module.

15.203.5.78 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to this module.

15.203.5.79 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.203.5.80 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

_out	The stream to print the assignment on.

15.203.5.81 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const  [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.203.5.82 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

_out	The stream to print the assignment on.

15.203.5.83 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const  [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.203.5.84 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const  [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.203.5.85 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.

15.203.5.86 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

15.203.5.87 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

15.203.5.88 receivedVariable()

`bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

15.203.5.89 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

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

15.203.5.90 removeCore()

`template<class 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

_subformula	The position of the subformula to remove.

Reimplemented from smtrat::Module.

15.203.5.91 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.203.5.92 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

15.203.5.93 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline],
[inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.203.5.94 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.203.5.95 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( ) [inline],
[protected], [virtual], [inherited]
```

Reimplemented in smtrat::PModule.

15.203.5.96 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

_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 in smtrat::PModule.

15.203.5.97 setId()

```
void smtrat::Module::setId ( std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.203.5.98 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.

15.203.5.99 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.

15.203.5.100 splitUnequalConstraint()

```
void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]
```

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.203.5.101 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.203.5.102 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 >.

15.203.5.103 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.203.5.104 updateModel() `template<class 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.

15.203.5.105 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).

15.203.6 Field Documentation

15.203.6.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.203.6.2 mAllModels

`std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.203.6.3 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.

15.203.6.4 mConstraintsToInform

`carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.203.6.5 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

15.203.6.6 mFirstPosInLastBranches

`std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.203.6.7 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.

15.203.6.8 mFirstUncheckedReceivedSubformula

`ModuleInput::const_iterator smtrat::Module::m↩FirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

15.203.6.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.203.6.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.203.6.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.203.6.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.203.6.13 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.

15.203.6.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.203.6.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.203.6.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.203.6.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.203.6.18 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.

15.203.6.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplitting↩Variables [static], [inherited]`

Reusable splitting variables.

15.203.6.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.203.6.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.203.6.22 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.

15.203.6.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

15.203.6.24 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).

15.203.6.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.204 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 ()

15.204.1 Detailed Description

Provides a simple implementation for Fourier Motzkin variable elimination for linear, existentially quantified constraints.

15.204.2 Member Typedef Documentation

15.204.2.1 FormulaPartition `using smtrat::qe::fm::FourierMotzkinQE::FormulaPartition = std↩ ::vector<std::vector<FormulaT> >`

15.204.3 Constructor & Destructor Documentation

15.204.3.1 FourierMotzkinQE()

```
smtrat::qe::fm::FourierMotzkinQE::FourierMotzkinQE (
    const FormulaT & qfree,
    const QEQuery & quantifiers ) [inline]
```

15.204.4 Member Function Documentation

15.204.4.1 eliminateQuantifiers()

```
FormulaT smtrat::qe::fm::FourierMotzkinQE::eliminateQuantifiers
( )
```

15.205 smtrat::FPPModule< Settings > Class Template Reference

```
#include <FPPModule.h>
```

Public Types

- typedef Settings SettingsType
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::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 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
- 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 _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.

- 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 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 &_branching↩Value) 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 _preferLeft↩Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩::vector< FormulaT >())

- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩CaseWeak=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.

15.205.1 Member Typedef Documentation

15.205.1.1 SettingsType

```
template<typename Settings >
typedef Settings smtrat::FPPModule< Settings >::SettingsType
```

15.205.2 Member Enumeration Documentation

15.205.2.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

15.205.3 Constructor & Destructor Documentation

15.205.3.1 FPPModule()

```
template<class Settings >
smtrat::FPPModule< Settings >::FPPModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = NULL )
```

15.205.3.2 ~FPPModule()

```
template<class Settings >
smtrat::FPPModule< Settings >::~FPPModule ( )
```

15.205.4 Member Function Documentation

15.205.4.1 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

_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.

15.205.4.2 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 >.

15.205.4.3 addCore()

```
template<class 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

_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.

15.205.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.205.4.5 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.

15.205.4.6 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.

15.205.4.7 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.

15.205.4.8 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.

15.205.4.9 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.

15.205.4.10 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

_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.

15.205.4.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const  [inline],
[inherited]
```

Returns All satisfying assignments, if existent.

15.205.4.12 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.

15.205.4.13 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.

15.205.4.14 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.

15.205.4.15 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.

15.205.4.16 branchAt() [1/6]

```
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]
```

15.205.4.17 branchAt() [2/6]

```
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]
```

15.205.4.18 branchAt() [3/6]

```
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]
```

15.205.4.19 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.205.4.20 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
    const typename P::PolyType & _poly,
    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]
```

15.205.4.21 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false )  [inline], [protected], [inherited]
```

15.205.4.22 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

15.205.4.23 checkCore()

```
template<class 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.

15.205.4.24 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
            std::vector< FormulaSetT >::const_iterator _infsubset,
            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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.205.4.25 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.

15.205.4.26 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.205.4.27 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.205.4.28 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.205.4.29 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.205.4.30 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.205.4.31 collectOrigins() [1/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

15.205.4.32 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.

15.205.4.33 collectTheoryPropagations()

`void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.205.4.34 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ToInform ( ) const [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.205.4.35 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

15.205.4.36 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)

15.205.4.37 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

_constraint	The constraint to remove from internal data structures.

15.205.4.38 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.205.4.39 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.205.4.40 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.205.4.41 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceived↩
VariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.205.4.42 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.205.4.43 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.

15.205.4.44 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.

15.205.4.45 freeSplittingVariable()

```
static void smtrat::Module::freeSplittingVariable (
    const FormulaT & _splittingVariable ) [inline], [static], [inherited]
```

15.205.4.46 generateTrivialInfeasibleSubset()

```
void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]
```

Stores the trivial infeasible subset being the set of received formulas.

15.205.4.47 getBackendsAllModels()

```
void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]
```

Stores all models of a backend in the list of all models of this module.

15.205.4.48 getBackendsModel()

```
void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]
```

15.205.4.49 getInfeasibleSubsets() [1/2]

```
void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]
```

Copies the infeasible subsets of the passed formula.

15.205.4.50 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.205.4.51 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.205.4.52 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.

15.205.4.53 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.205.4.54 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.205.4.55 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.205.4.56 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 reveived formula, if the Boolean is true.

Reimplemented from smtrat::Module.

15.205.4.57 hasLemmas() bool smtrat::Module::hasLemmas ( ) [inline], [inherited]

Checks whether this module or any of its backends provides any lemmas.

15.205.4.58 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).

15.205.4.59 id() std::size_t smtrat::Module::id ( ) const [inline], [inherited]

Returns A unique ID to identify this module instance.

15.205.4.60 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.

15.205.4.61 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.205.4.62 informBackends()

```
void smtrat::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.

15.205.4.63 informCore()

```
template<class 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

_constraint	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.

15.205.4.64 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.205.4.65 init()

```
template<class 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.

15.205.4.66 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.205.4.67 isLemmaLevel() `bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.205.4.68 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().

15.205.4.69 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.

15.205.4.70 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.205.4.71 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.205.4.72 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.

15.205.4.73 moduleName()

```
virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]
```

Returns The name of the given module type as name.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::EQPreprocessingModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::VSModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::GBModule< Settings >, smtrat::IntEqModule< Setti smtrat::CubeLIAModule< Settings >, smtrat::CurryModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IncWidthModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::BEModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::SplitSOSModule< Settings >.

15.205.4.74 objective()

```
carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]
```

15.205.4.75 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
    const FormulaT & _origin ) const [protected], [inherited]
```

15.205.4.76 passedFormulaBegin()

```
ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]
```

Returns An iterator to the end of the passed formula. TODO: disable this method

15.205.4.77 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula.

15.205.4.78 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to the backends of this module.

15.205.4.79 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to this module.

15.205.4.80 print()

```
void smtrat::Module::print (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.205.4.81 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

_out	The stream to print the assignment on.

15.205.4.82 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.205.4.83 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

_out	The stream to print the assignment on.

15.205.4.84 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.205.4.85 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.205.4.86 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.

15.205.4.87 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

15.205.4.88 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

15.205.4.89 receivedVariable()

`bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

15.205.4.90 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

_subformula	The sub formula of the received formula to remove.

Reimplemented from smtrat::Module.

15.205.4.91 removeCore()

`template<class Settings > void smtrat::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

_subformula	The position of the subformula to remove.

Reimplemented from smtrat::Module.

15.205.4.92 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.205.4.93 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]
```

15.205.4.94 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.205.4.95 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.205.4.96 runBackends() [1/2]

```
virtual Answer smtrat::PModule::runBackends ( ) [inline], [virtual], [inherited]
```

Reimplemented from smtrat::Module.

15.205.4.97 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.

15.205.4.98 setId() `void smtrat::Module::setId ( std::size_t _id ) [inline], [inherited]`

Sets this modules unique ID to identify itself.

Parameters

↩ _id	The id to set this module's id to.

15.205.4.99 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.

15.205.4.100 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.

15.205.4.101 splitUnequalConstraint() `void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.205.4.102 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.205.4.103 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 >.

15.205.4.104 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.205.4.105 updateModel() `template<class Settings > void smtrat::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 smtrat::Module.

15.205.4.106 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).

15.205.5 Field Documentation

15.205.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.205.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.205.5.3 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.

15.205.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.205.5.5 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

15.205.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.205.5.7 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.

15.205.5.8 mFirstUncheckedReceivedSubformula `ModuleInput::const_iterator smtrat::Module::m↩ FirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

15.205.5.9 mFoundAnswer `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.205.5.10 mFullCheck `bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.205.5.11 mInfeasibleSubsets `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.205.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.205.5.13 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.

15.205.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.205.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.205.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.205.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.205.5.18 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.

15.205.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]`

Reusable splitting variables.

15.205.5.20 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.205.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.205.5.22 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.

15.205.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

15.205.5.24 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).

15.205.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.206 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.

15.206.1 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().

15.206.2 Constructor & Destructor Documentation

15.206.2.1 freelist() [1/2] template<typename T>

```
smtrat::freelist< T >::freelist ( ) [inline], [explicit]
```

15.206.2.2 freelist() [2/2] template<typename T>

```
smtrat::freelist< T >::freelist (
            std::size_t chunk_size_ ) [inline], [explicit]
```

15.206.2.3 ~freelist() template<typename T>

```
smtrat::freelist< T >::~freelist ( ) [inline]
```

15.206.3 Member Function Documentation

15.206.3.1 alloc() template<typename T>

```
void* smtrat::freelist< T >::alloc ( ) [inline]
```

Allocate an object of size 1<<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.

15.206.3.2 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.

15.206.3.3 emplace() template<typename T>

```
template<typename... Args>
T& smtrat::freelist< T >::emplace (
            Args &&... args ) [inline]
```

15.206.3.4 free() template<typename T>

```
void smtrat::freelist< T >::free (
            void * ptr )  [inline]
```

Freeing means just prepending the free'd element to the front of the freelist.

15.207 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>
```

15.207.1 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.

15.208 smtrat::cad::FullSampleComparator< Iterator, Strategy > Struct Template Reference

```
#include <SampleComparator.h>
```

15.209 smtrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::T > Struct Template Reference

```
#include <SampleComparator.h>
```

15.210 smtrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::Type > Struct Template Reference

```
#include <SampleComparator.h>
```

15.211 smtrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::Value > Struct Template Reference

```
#include <SampleComparator.h>
```

15.212 smtrat::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

15.212.1 Constructor & Destructor Documentation

15.212.1.1 ~FunctionInstantiator()

`virtual smtrat::parser::FunctionInstantiator::~FunctionInstantiator ( ) [inline], [virtual]`

15.212.2 Member Function Documentation

15.212.2.1 convert() [1/2]

```
template<typename T >
bool smtrat::parser::FunctionInstantiator::convert (
            const std::vector< types::TermType > & from,
            std::vector< T > & to ) const [inline]
```

15.212.2.2 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]
```

15.212.2.3 operator()()

```
virtual bool smtrat::parser::FunctionInstantiator::operator() (
            const std::vector< types::TermType > & ,
            types::TermType & ,
            TheoryError & errors ) const [inline], [virtual]
```

15.213 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 LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- GBModule (const ModuleInput *const, Conditionals &, Manager *const =NULL)
- virtual ~GBModule ()
- bool addCore (ModuleInput::const_iterator _formula)

  Adds the constraint to the known constraints of the module.
- virtual Answer checkCore ()

  A theory call to the GBModule.
- void removeCore (ModuleInput::const_iterator _formula)

  Removes the constraint from the GBModule.
- void printStateHistory ()
- void printRewriteRules ()
- 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 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 _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

- bool constraintByGB (carl::Relation cr)
- void pushBacktrackPoint (ModuleInput::const_iterator btpoint)

  To implement backtrackability, we save the current state after each equality we add.
- void popBacktrackPoint (ModuleInput::const_iterator btpoint)

  Pops all states from the stack until the state which we had before the constraint was added.
- bool saveState ()
- FormulasT generateReasons (const carl::BitVector &reasons)

  Generate reason sets from reason vectors.
- FormulaSetT generateReasonSet (const carl::BitVector &reasons)

  Generate reason sets from reason vectors.
- void passGB ()

  Add the equalities from the Groebner basis to the passed formula.

- 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)

  Transforms a given inequality to a polynomial such that p = 0 is equivalent to the given constraint.
- GBPolynomial callGroebnerToSDP (const Ideal &gb)
- bool iterativeVariableRewriting ()

  With the new groebner basis, we search for radical-members which then can be added to the GB.
- bool findTrivialFactorisations ()

  A method which looks for polynomials which have trivial factors.
- void processNewConstraint (ModuleInput::const_iterator _formula)

  Method which updates internal data structures to reflect the added formula.
- void handleConstraintToGBQueue (ModuleInput::const_iterator _formula)

  Method which adds constraint to the GB-process.
- void handleConstraintNotToGB (ModuleInput::const_iterator _formula)

  Method which adds constraints to our internal datastructures without adding them to the GB.
- void removeReceivedFormulaFromNewInequalities (ModuleInput::const_iterator _formula)

  Removes a received formula from the list of new inequalities.
- void removeSubformulaFromPassedFormula (ModuleInput::iterator _formula)

  This function is overwritten such that it is visible to the InequalitiesTable.
- GBPolynomial rewriteVariable (const GBPolynomial &, const carl::Variable &, const TermT &, const BitVector &)
- bool validityCheck ()

  A validity check of the data structures which can be used to assert valid behaviour.
- 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 &_branching↩ Value) 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 >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, 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

- 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< size_t, 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 >

15.213.1 Detailed Description

template<class Settings>
class smtrat::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

15.213.2 Member Typedef Documentation

15.213.2.1 GBPolynomial template<class Settings>

```
typedef Settings::PolynomialWithReasons smtrat::GBModule< Settings >::GBPolynomial
```

15.213.2.2 Ideal template<class Settings>

```
typedef Settings::MultivariateIdeal smtrat::GBModule< Settings >::Ideal
```

15.213.2.3 Order template<class Settings>

```
typedef Settings::Order smtrat::GBModule< Settings >::Order
```

15.213.2.4 SettingsType template<class Settings>

```
typedef Settings smtrat::GBModule< Settings >::SettingsType
```

15.213.3 Member Enumeration Documentation

15.213.3.1 LemmaType enum smtrat::Module::LemmaType : unsigned [strong], [inherited]

Enumerator

NORMAL	
PERMANENT	

15.213.4 Constructor & Destructor Documentation

15.213.4.1 GBModule() template<class Settings >

```
smtrat::GBModule< Settings >::GBModule (
            const ModuleInput * const _formula,
            Conditionals & _conditionals,
            Manager * const _manager = NULL )
```

15.213.4.2 ~GBModule() template<class Settings >

```
smtrat::GBModule< Settings >::~GBModule ( ) [virtual]
```

15.213.5 Member Function Documentation

15.213.5.1 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.

15.213.5.2 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

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

15.213.5.3 addCore()

```
template<class Settings >
bool smtrat::GBModule< Settings >::addCore (
    ModuleInput::const_iterator _formula ) [virtual]
```

Adds the constraint to the known constraints of the module.

This includes scanning variables as well as transforming inequalities, if this is enabled.

Parameters

_formula	A REALCONSTRAINT which should be regarded by the next theory call.

Returns true

Reimplemented from smtrat::Module.

15.213.5.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.213.5.5 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.

15.213.5.6 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.

15.213.5.7 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.

15.213.5.8 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.

15.213.5.9 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

_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.

15.213.5.10 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

_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.

15.213.5.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const  [inline],
[inherited]
```

Returns All satisfying assignments, if existent.

15.213.5.12 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.

15.213.5.13 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.

15.213.5.14 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.

15.213.5.15 branchAt() [1/6]

```
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]
```

15.213.5.16 branchAt() [2/6]

```
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]
```

15.213.5.17 branchAt() [3/6]

```
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]
```

15.213.5.18 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.213.5.19 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
    const typename P::PolyType & _poly,
    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]
```

15.213.5.20 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
    const typename P::PolyType & _poly,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

15.213.5.21 callGroebnerToSDP()

```
template<class Settings>
GBPolynomial smtrat::GBModule< Settings >::callGroebnerToSDP (
            const Ideal & gb ) [protected]
```

15.213.5.22 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in smtrat::PModule.

15.213.5.23 checkCore()

```
template<class Settings >
Answer smtrat::GBModule< Settings >::checkCore ( ) [virtual]
```

A theory call to the GBModule.

The exact working of this module depends on the settings in GBSettings.

Returns (TRUE,FALSE,UNKNOWN) dependent on the asserted constraints.

TODO fix this

Reimplemented from smtrat::Module.

15.213.5.24 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
            std::vector< FormulaSetT >::const_iterator _infsubset,
            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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.213.5.25 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.

15.213.5.26 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.213.5.27 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.213.5.28 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.213.5.29 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.213.5.30 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.213.5.31 collectOrigins() [1/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

15.213.5.32 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.

15.213.5.33 collectTheoryPropagations()

`void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.213.5.34 constraintByGB()

```
template<class Settings >
bool smtrat::GBModule< Settings >::constraintByGB (
            carl::Relation cr )  [protected]
```

15.213.5.35 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const  [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.213.5.36 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::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.213.5.37 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)

15.213.5.38 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

_constraint	The constraint to remove from internal data structures.

15.213.5.39 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.213.5.40 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.213.5.41 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.213.5.42 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.213.5.43 factorisedConstraintDeduction()

```
template<class Settings>
void smtrat::GBModule< Settings >::factorisedConstraintDeduction (
            const std::list< GBPolynomial > & factorisation,
            const carl::BitVector & reasons ) [protected]
```

15.213.5.44 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
            const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.213.5.45 findTrivialFactorisations()

```
template<class Settings >
bool smtrat::GBModule< Settings >::findTrivialFactorisations ( ) [protected]
```

A method which looks for polynomials which have trivial factors.

15.213.5.46 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.

15.213.5.47 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.

15.213.5.48 freeSplittingVariable()

```
static void smtrat::Module::freeSplittingVariable (
            const FormulaT & _splittingVariable ) [inline], [static], [inherited]
```

15.213.5.49 generateReasons()

```
template<class Settings >
FormulasT smtrat::GBModule< Settings >::generateReasons (
            const carl::BitVector & reasons ) [protected]
```

Generate reason sets from reason vectors.

Parameters

reasons	The reasons vector.

Returns The reason set.

15.213.5.50 generateReasonSet()

```
template<class Settings >
FormulaSetT smtrat::GBModule< Settings >::generateReasonSet (
            const carl::BitVector & reasons ) [protected]
```

Generate reason sets from reason vectors.

Parameters

reasons	The reasons vector.

Returns The reason set.

15.213.5.51 generateTrivialInfeasibleSubset()

```
void smtrat::Module::generateTrivialInfeasibleSubset
( ) [inline], [protected], [inherited]
```

Stores the trivial infeasible subset being the set of received formulas.

15.213.5.52 getBackendsAllModels()

```
void smtrat::Module::getBackendsAllModels ( ) const [protected],
[inherited]
```

Stores all models of a backend in the list of all models of this module.

15.213.5.53 getBackendsModel()

```
void smtrat::Module::getBackendsModel ( ) const [protected],
[inherited]
```

15.213.5.54 getInfeasibleSubsets() [1/2]

```
void smtrat::Module::getInfeasibleSubsets ( ) [protected],
[inherited]
```

Copies the infeasible subsets of the passed formula.

15.213.5.55 getInfeasibleSubsets() [2/2]

```
std::vector< FormulaSetT > smtrat::Module::getInfeasible↩
Subsets (
            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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.213.5.56 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.213.5.57 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.

15.213.5.58 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.213.5.59 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.213.5.60 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.

15.213.5.61 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 reveived formula, if the Boolean is true.

Reimplemented in smtrat::PModule.

15.213.5.62 handleConstraintNotToGB()

```
template<class Settings >
void smtrat::GBModule< Settings >::handleConstraintNotToGB (
            ModuleInput::const_iterator _formula ) [protected]
```

Method which adds constraints to our internal datastructures without adding them to the GB.

E.g. inequalities are added to the inequalitiestable.

Parameters

_formula	

15.213.5.63 handleConstraintToGBQueue()

```
template<class Settings >
void smtrat::GBModule< Settings >::handleConstraintToGBQueue (
            ModuleInput::const_iterator _formula ) [protected]
```

Method which adds constraint to the GB-process.

Parameters

_formula	

15.213.5.64 hasLemmas() `bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

15.213.5.65 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).

15.213.5.66 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns A unique ID to identify this module instance.

15.213.5.67 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.

15.213.5.68 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.213.5.69 informBackends() `void smtrat::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.

15.213.5.70 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, and smtrat::ICPModule< ICPSettings1 >.

15.213.5.71 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.

15.213.5.72 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::FPPModule< Settings >.

15.213.5.73 is_minimizing()

`bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.213.5.74 isLemmaLevel()

`bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.213.5.75 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().

15.213.5.76 iterativeVariableRewriting() `template<class Settings >`
`bool smtrat::GBModule< Settings >::iterativeVariableRewriting ( ) [protected]`

With the new groebner basis, we search for radical-members which then can be added to the GB.

Returns

TODO fix this

15.213.5.77 knownConstraintDeduction() `template<class Settings >`
```
void smtrat::GBModule< Settings >::knownConstraintDeduction (
            const std::list< std::pair< carl::BitVector, carl::BitVector > > & deductions )
[protected]
```

15.213.5.78 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.

15.213.5.79 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.213.5.80 model()

```
const Model& smtrat::Module::model ( ) const [inline], [inherited]
```

Returns The assignment of the current satisfiable result, if existent.

15.213.5.81 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.

15.213.5.82 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.

15.213.5.83 newConstraintDeduction()

```
template<class Settings >
void smtrat::GBModule< Settings >::newConstraintDeduction ( ) [protected]
```

15.213.5.84 objective()

```
carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]
```

15.213.5.85 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
    const FormulaT & _origin ) const [protected], [inherited]
```

15.213.5.86 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula. TODO: disable this method

15.213.5.87 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula.

15.213.5.88 passGB()

```
template<class Settings >
void smtrat::GBModule< Settings >::passGB ( ) [protected]
```

Add the equalities from the Groebner basis to the passed formula.

Adds the reason vector.

15.213.5.89 popBacktrackPoint()

```
template<class Settings >
void smtrat::GBModule< Settings >::popBacktrackPoint (
            ModuleInput::const_iterator btpoint ) [protected]
```

Pops all states from the stack until the state which we had before the constraint was added.

Then, we make new states with all equalities which were added afterwards.

Parameters

a	pointer in the received formula to the constraint which will be removed.

15.213.5.90 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to the backends of this module.

15.213.5.91 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to this module.

15.213.5.92 print()

```
void smtrat::Module::print (
            const std::string & _initiation = "***" ) const [inherited]
```

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.213.5.93 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

_out	The stream to print the assignment on.

15.213.5.94 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.213.5.95 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

_out	The stream to print the assignment on.

15.213.5.96 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.213.5.97 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const  [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.213.5.98 printRewriteRules()

```
template<class Settings>
void smtrat::GBModule< Settings >::printRewriteRules ( )
```

15.213.5.99 printStateHistory()

```
template<class Settings>
void smtrat::GBModule< Settings >::printStateHistory ( )
```

15.213.5.100 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.

15.213.5.101 processNewConstraint()

```
template<class Settings >
void smtrat::GBModule< Settings >::processNewConstraint (
    ModuleInput::const_iterator _formula )  [protected]
```

Method which updates internal data structures to reflect the added formula.

Parameters

_formula	

15.213.5.102 pushBacktrackPoint()

```
template<class Settings >
void smtrat::GBModule< Settings >::pushBacktrackPoint (
    ModuleInput::const_iterator btpoint )  [protected]
```

To implement backtrackability, we save the current state after each equality we add.

Parameters

btpoint	The equality we have removed

15.213.5.103 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

15.213.5.104 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

15.213.5.105 receivedVariable()

`bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

15.213.5.106 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.

15.213.5.107 removeCore()

`template<class Settings > void smtrat::GBModule< Settings >::removeCore ( ModuleInput::const_iterator _formula ) [virtual]`

Removes the constraint from the GBModule.

Notice: Whenever a constraint is removed which was not asserted before, this leads to an unwanted error. A general approach for this has to be found.

Parameters

_formula	the constraint which should be removed.

Reimplemented from smtrat::Module.

15.213.5.108 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin (
            ModuleInput::iterator _formula,
            const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.213.5.109 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

15.213.5.110 removeReceivedFormulaFromNewInequalities()

```
template<class Settings >
void smtrat::GBModule< Settings >::removeReceivedFormulaFromNewInequalities (
            ModuleInput::const_iterator _formula ) [protected]
```

Removes a received formula from the list of new inequalities.

It assumes that there is only one such element in the list.

Parameters

_formula	

15.213.5.111 removeSubformulaFromPassedFormula()

```
template<class Settings >
void smtrat::GBModule< Settings >::removeSubformulaFromPassedFormula (
            ModuleInput::iterator _formula ) [protected]
```

This function is overwritten such that it is visible to the InequalitiesTable.

For more details take a look at Module::eraseSubformulaFromPassedFormula(..)

Parameters

_formula	

15.213.5.112 rewriteVariable()

```
template<class Settings>
GBPolynomial smtrat::GBModule< Settings >::rewriteVariable (
            const GBPolynomial & ,
            const carl::Variable & ,
            const TermT & ,
            const BitVector & ) [inline], [protected]
```

15.213.5.113 rPassedFormula() `const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]`

Returns A reference to the formula passed to the backends of this module.

15.213.5.114 rReceivedFormula() `const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]`

Returns A reference to the formula passed to this module.

15.213.5.115 runBackends() [1/2] `virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]`

Reimplemented in smtrat::PModule.

15.213.5.116 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

_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 in smtrat::PModule.

15.213.5.117 saveState()

```
template<class Settings >
bool smtrat::GBModule< Settings >::saveState ( ) [protected]
```

Returns

15.213.5.118 setId()

```
void smtrat::Module::setId (
    std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ _↩ *id*	The id to set this module's id to.

15.213.5.119 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.

15.213.5.120 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.

15.213.5.121 splitUnequalConstraint()

`void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.213.5.122 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.213.5.123 transformIntoEquality() template<class Settings >

GBModule< Settings >::GBPolynomial smtrat::GBModule< Settings >::transformIntoEquality (ModuleInput::const_iterator *constraint*) [protected]

Transforms a given inequality to a polynomial such that p = 0 is equivalent to the given constraint.

This is done by inserting an additional variable which has an index which is given by the id of the given inequality.

Parameters

constraint	a pointer to the inequality

Returns The polynomial which represents the equality.

15.213.5.124 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 >.

15.213.5.125 updateLemmas() void smtrat::Module::updateLemmas ( ) [inherited]

Stores all lemmas of any backend of this module in its own lemma vector.

15.213.5.126 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::SATModule< Settings >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRAS smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::EQPreprocessingModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::PModule, smtrat::VSModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, smtrat::LVEModule< Settings >, smtrat::MCBModule< Settings >, smtrat::ESModule< Settings >, and smtrat::GBPPModule< Settings >.

15.213.5.127 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).

15.213.5.128 validityCheck() template<class Settings >

bool smtrat::GBModule< Settings >::validityCheck ( ) [protected]

A validity check of the data structures which can be used to assert valid behaviour.

Returns true, iff the backtrackpoints are valid.

15.213.6 Friends And Related Function Documentation

15.213.6.1 InequalitiesTable< Settings > template<class Settings>

friend class InequalitiesTable< Settings > [friend]

15.213.7 Field Documentation

15.213.7.1 mAdditionalVarMap template<class Settings>

std::map<size_t, carl::Variable> smtrat::GBModule< Settings >::mAdditionalVarMap [protected]

15.213.7.2 mAllBackends std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]

The backends of this module which have been used.

15.213.7.3 mAllModels std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]

Stores all satisfying assignments.

15.213.7.4 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.

15.213.7.5 mBacktrackPoints template<class Settings>

std::vector<ModuleInput::const_iterator> smtrat::GBModule< Settings >::mBacktrackPoints [protected]

The vector of backtrack points, which has pointers to received constraints.

15.213.7.6 mBasis template<class Settings>

Settings::Groebner smtrat::GBModule< Settings >::mBasis [protected]

The current Groebner basis.

15.213.7.7 mConstraintsToInform carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]

Stores the constraints which the backends must be informed about.

15.213.7.8 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

15.213.7.9 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.213.7.10 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.

15.213.7.11 mFirstUncheckedReceivedSubformula `ModuleInput::const_iterator smtrat::Module::m↩ FirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

15.213.7.12 mFoundAnswer `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.213.7.13 mFullCheck `bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.213.7.14 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)

15.213.7.15 mInequalities `template<class Settings> InequalitiesTable<Settings> smtrat::GBModule< Settings >::mInequalities [protected]`

The inequalities table for handling inequalities.

15.213.7.16 mInfeasibleSubsets `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.213.7.17 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.213.7.18 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.

15.213.7.19 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.213.7.20 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.213.7.21 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.213.7.22 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.

15.213.7.23 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.213.7.24 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.

15.213.7.25 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

15.213.7.26 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.213.7.27 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.

15.213.7.28 mRewriteRules

`template<class Settings> groebner::RewriteRules smtrat::GBModule< Settings >::mRewriteRules [protected]`

The rewrite rules for the variables.

15.213.7.29 mSmallerMusesCheckCounter

`unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.213.7.30 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.

15.213.7.31 mStateHistory

`template<class Settings> std::list<GBModuleState<Settings> > smtrat::GBModule< Settings >::mStateHistory [protected]`

Saves the relevant history to support backtracking.

15.213.7.32 mTheoryPropagations

`std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

15.213.7.33 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).

15.213.7.34 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.214 smtrat::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

15.214.1 Detailed Description

template<typename Settings>
class smtrat::GBModuleState< Settings >

A class to save the current state of a GBModule.

Used for backtracking-support

15.214.2 Constructor & Destructor Documentation

15.214.2.1 GBModuleState() [1/2] `template<typename Settings > smtrat::GBModuleState< Settings >::GBModuleState ( ) [inline]`

15.214.2.2 GBModuleState() [2/2] `template<typename Settings > smtrat::GBModuleState< Settings >::GBModuleState ( const typename Settings::Groebner & basisCalculation, const std::map< carl::Variable, std::pair< TermT, carl::BitVector > > & rewrites ) [inline]`

15.214.3 Member Function Documentation

15.214.3.1 getBasis() `template<typename Settings > const Settings::Groebner& smtrat::GBModuleState< Settings >::getBasis ( ) const [inline]`

15.214.3.2 getRewriteRules() `template<typename Settings > const std::map<carl::Variable, std::pair<TermT, carl::BitVector> >& smtrat::GBModuleState< Settings >::getRewriteRules ( ) const [inline]`

15.214.4 Field Documentation

15.214.4.1 mBasis `template<typename Settings >`

`const Settings::Groebner smtrat::GBModuleState< Settings >::mBasis [protected]`

The state of the basis.

15.214.4.2 mRewrites `template<typename Settings >`

`const std::map<carl::Variable, std::pair<TermT, carl::BitVector> > smtrat::GBModuleState< Settings >::mRewrites [protected]`

15.215 smtrat::GBPPModule< Settings > Class Template Reference

`#include <GBPPModule.h>`

Public Types

- typedef Settings SettingsType
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::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 ()
- 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 smtrat::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 _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.

- 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 &_branching↩Value) 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 _preferLeft↩Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩::vector< FormulaT >())

- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩CaseWeak=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.

15.215.1 Member Typedef Documentation

15.215.1.1 SettingsType

```
template<typename Settings >
typedef Settings smtrat::GBPPModule< Settings >::SettingsType
```

15.215.2 Member Enumeration Documentation

15.215.2.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

15.215.3 Constructor & Destructor Documentation

15.215.3.1 GBPPModule()

```
template<class Settings >
smtrat::GBPPModule< Settings >::GBPPModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

15.215.3.2 ~GBPPModule()

```
template<class Settings >
smtrat::GBPPModule< Settings >::~GBPPModule ( )
```

15.215.4 Member Function Documentation

15.215.4.1 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

_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.

15.215.4.2 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 >.

15.215.4.3 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

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 smtrat::SATModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::VSModule< Settings >, smtrat::PFEModule< Settings >, smtrat::EQModule< Setting smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Sett smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, and smtrat::GBModule< Settings >.

15.215.4.4 addInformationRelevantFormula()

`void smtrat::Module::addInformationRelevantFormula ( const FormulaT & formula ) [protected], [inherited]`

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.215.4.5 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.

15.215.4.6 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.

15.215.4.7 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.

15.215.4.8 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.

15.215.4.9 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.

15.215.4.10 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

_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.

15.215.4.11 allModels()

`const std::vector<Model>& smtrat::Module::allModels ( ) const [inline], [inherited]`

Returns All satisfying assignments, if existent.

15.215.4.12 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.

15.215.4.13 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.

15.215.4.14 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.

15.215.4.15 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.

15.215.4.16 branchAt() [1/6]

```
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]
```

15.215.4.17 branchAt() [2/6]

```
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]
```

15.215.4.18 branchAt() [3/6]

```
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]
```

15.215.4.19 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.215.4.20 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P
>> = carl::dummy>
bool smtrat::Module::branchAt (
    const typename P::PolyType & _poly,
```

```
            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]
```

15.215.4.21 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P
>> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false )  [inline], [protected], [inherited]
```

15.215.4.22 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

15.215.4.23 checkCore()

```
template<class 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.

15.215.4.24 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
    std::vector< FormulaSetT >::const_iterator _infsubset,
    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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.215.4.25 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.

15.215.4.26 cleanModel()

```
void smtrat::Module::cleanModel ( ) const [inline], [protected],
[inherited]
```

Substitutes variable occurrences by its model value in the model values of other variables.

15.215.4.27 clearLemmas()

```
void smtrat::Module::clearLemmas ( ) [inline], [inherited]
```

Deletes all yet found lemmas.

15.215.4.28 clearModel()

```
void smtrat::Module::clearModel ( ) const [inline], [protected],
[inherited]
```

Clears the assignment, if any was found.

15.215.4.29 clearModels()

```
void smtrat::Module::clearModels ( ) const [inline], [protected],
[inherited]
```

Clears all assignments, if any was found.

15.215.4.30 clearPassedFormula()

```
void smtrat::Module::clearPassedFormula ( ) [protected],
[inherited]
```

15.215.4.31 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

15.215.4.32 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.

15.215.4.33 collectTheoryPropagations()

```
void smtrat::Module::collectTheoryPropagations ( ) [inherited]
```

15.215.4.34 constraintsToInform()

```
const carl::FastSet<FormulaT>& smtrat::Module::constraints↩
ToInform ( ) const [inline], [inherited]
```

Returns The constraints which the backends must be informed about.

15.215.4.35 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

15.215.4.36 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)

15.215.4.37 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

_constraint	The constraint to remove from internal data structures.

15.215.4.38 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.215.4.39 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.215.4.40 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.215.4.41 excludeNotReceivedVariablesFromModel()

`void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

15.215.4.42 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.215.4.43 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.

15.215.4.44 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.

15.215.4.45 freeSplittingVariable()

```
static void smtrat::Module::freeSplittingVariable (
    const FormulaT & _splittingVariable ) [inline], [static], [inherited]
```

15.215.4.46 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

15.215.4.47 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

15.215.4.48 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

15.215.4.49 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

15.215.4.50 getInfeasibleSubsets() [2/2] `std::vector< FormulaSetT > smtrat::Module::getInfeasible↩ Subsets ( 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.215.4.51 getInformationRelevantFormulas() `const std::set< FormulaT > & smtrat::Module::get↩ InformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.215.4.52 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.

15.215.4.53 getOrigins() [1/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

15.215.4.54 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.215.4.55 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.215.4.56 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 reveived formula, if the Boolean is true.

Reimplemented from smtrat::Module.

15.215.4.57 hasLemmas()

```
bool smtrat::Module::hasLemmas ( ) [inline], [inherited]
```

Checks whether this module or any of its backends provides any lemmas.

15.215.4.58 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).

15.215.4.59 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns A unique ID to identify this module instance.

15.215.4.60 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.

15.215.4.61 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.215.4.62 informBackends() `void smtrat::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.

15.215.4.63 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, and smtrat::ICPModule< ICPSettings1 >.

15.215.4.64 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.215.4.65 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::FPPModule< Settings >.

15.215.4.66 is_minimizing()

```
bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]
```

15.215.4.67 isLemmaLevel()

```
bool smtrat::Module::isLemmaLevel (
    LemmaLevel level ) [protected], [inherited]
```

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.215.4.68 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().

15.215.4.69 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.

15.215.4.70 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.215.4.71 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.215.4.72 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.

15.215.4.73 moduleName()

`virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]`

Returns The name of the given module type as name.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::EQPreprocessingModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::VSModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::GBModule< Settings >, smtrat::IntEqModule< Setti smtrat::CubeLIAModule< Settings >, smtrat::CurryModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IncWidthModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::BEModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::SplitSOSModule< Settings >.

15.215.4.74 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

15.215.4.75 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

15.215.4.76 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula. TODO: disable this method

15.215.4.77 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula.

15.215.4.78 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to the backends of this module.

15.215.4.79 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to this module.

15.215.4.80 print()

```
void smtrat::Module::print (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.215.4.81 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

_out	The stream to print the assignment on.

15.215.4.82 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.215.4.83 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

_out	The stream to print the assignment on.

15.215.4.84 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.215.4.85 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.215.4.86 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.

15.215.4.87 receivedFormulaChecked()

```
void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]
```

Notifies that the received formulas has been checked.

15.215.4.88 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩InfeasibleSubset (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

15.215.4.89 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const [inline], [inherited]
```

15.215.4.90 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

_subformula	The sub formula of the received formula to remove.

Reimplemented from smtrat::Module.

15.215.4.91 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

formula	The sub formula of the received formula to remove.

Reimplemented in smtrat::SATModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::VSModule< Settings >, smtrat::PFEModule< Settings >, smtrat::EQModule< Setting smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Sett smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, and smtrat::GBModule< Settings >.

15.215.4.92 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin
(
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.215.4.93 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

15.215.4.94 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline],
[inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.215.4.95 rReceivedFormula() `const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]`

Returns A reference to the formula passed to this module.

15.215.4.96 runBackends() [1/2] `virtual Answer smtrat::PModule::runBackends ( ) [inline], [virtual], [inherited]`

Reimplemented from smtrat::Module.

15.215.4.97 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.

15.215.4.98 setId()

```
void smtrat::Module::setId (
    std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.215.4.99 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.

15.215.4.100 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.

15.215.4.101 splitUnequalConstraint()

`void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.215.4.102 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.215.4.103 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 >.

15.215.4.104 updateLemmas()

`void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.215.4.105 updateModel() template<class 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.

15.215.4.106 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).

15.215.5 Field Documentation

15.215.5.1 mAllBackends std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]

The backends of this module which have been used.

15.215.5.2 mAllModels std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]

Stores all satisfying assignments.

15.215.5.3 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.

15.215.5.4 mConstraintsToInform carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]

Stores the constraints which the backends must be informed about.

15.215.5.5 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

15.215.5.6 mFirstPosInLastBranches std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]

The beginning of the cyclic buffer storing the last branches.

15.215.5.7 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.

15.215.5.8 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.

15.215.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.215.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.215.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.215.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.215.5.13 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.

15.215.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.215.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.215.5.16 mModelComputed `bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.215.5.17 mNumOfBranchVarsToStore `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.215.5.18 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.

15.215.5.19 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

15.215.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.215.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.215.5.22 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.

15.215.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

15.215.5.24 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).

15.215.5.25 mVariableCounters

`std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.216 smtrat::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

15.216.1 Member Function Documentation

15.216.1.1 operator()()

```
template<typename Settings>
std::size_t smtrat::EQModule< Settings >::not_asserted_equality::hash::operator() (
            const not_asserted_equality & nae ) const [inline], [noexcept]
```

15.217 Minisat::Hash< K > Struct Template Reference

`#include <Map.h>`

Public Member Functions

- uint32_t operator() (const K &k) const

15.217.1 Member Function Documentation

15.217.1.1 operator()()

```
template<class K >
uint32_t Minisat::Hash< K >::operator() (
            const K & k ) const [inline]
```

15.218 std::hash< const smtrat::expression::ExpressionContent * > Struct Template Reference

`#include <ExpressionHash.h>`

Public Member Functions

- std::size_t operator() (const smtrat::expression::ExpressionContent *ec) const

15.218.1 Member Function Documentation

15.218.1.1 operator()()

```
std::size_t std::hash< const smtrat::expression::ExpressionContent * >↩
::operator() (
            const smtrat::expression::ExpressionContent * ec ) const [inline]
```

15.219 std::hash< smtrat::expression::BinaryExpression > Struct Template Reference

`#include <ExpressionHash.h>`

Public Member Functions

- std::size_t operator() (const smtrat::expression::BinaryExpression &ec) const

15.219.1 Member Function Documentation

15.219.1.1 operator()()

```
std::size_t std::hash< smtrat::expression::BinaryExpression >::operator()
(
            const smtrat::expression::BinaryExpression & ec ) const [inline]
```

15.220 std::hash< smtrat::expression::Expression > Struct Template Reference

```
#include <ExpressionHash.h>
```

Public Member Functions

- std::size_t operator() (const smtrat::expression::Expression &expr) const

15.220.1 Member Function Documentation

15.220.1.1 operator()()

```
std::size_t std::hash< smtrat::expression::Expression >::operator() (
    const smtrat::expression::Expression & expr ) const [inline]
```

15.221 std::hash< smtrat::expression::ITEExpression > Struct Template Reference

```
#include <ExpressionHash.h>
```

Public Member Functions

- std::size_t operator() (const smtrat::expression::ITEExpression &ec) const

15.221.1 Member Function Documentation

15.221.1.1 operator()()

```
std::size_t std::hash< smtrat::expression::ITEExpression >::operator()
(
    const smtrat::expression::ITEExpression & ec ) const [inline]
```

15.222 std::hash< smtrat::expression::NaryExpression > Struct Template Reference

```
#include <ExpressionHash.h>
```

Public Member Functions

- std::size_t operator() (const smtrat::expression::NaryExpression &ec) const

15.222.1 Member Function Documentation

15.222.1.1 operator()()

```
std::size_t std::hash< smtrat::expression::NaryExpression >::operator()
(
    const smtrat::expression::NaryExpression & ec ) const [inline]
```

15.223 std::hash< smtrat::expression::QuantifierExpression > Struct Template Reference

```
#include <ExpressionHash.h>
```

Public Member Functions

- std::size_t operator() (const smtrat::expression::QuantifierExpression &ec) const

15.223.1 Member Function Documentation

15.223.1.1 operator()()

```
std::size_t std::hash< smtrat::expression::QuantifierExpression >↩
::operator() (
    const smtrat::expression::QuantifierExpression & ec ) const [inline]
```

15.224 std::hash< smtrat::expression::UnaryExpression > Struct Template Reference

```
#include <ExpressionHash.h>
```

Public Member Functions

- std::size_t operator() (const smtrat::expression::UnaryExpression &ec) const

15.224.1 Member Function Documentation

15.224.1.1 operator()()

```
std::size_t std::hash< smtrat::expression::UnaryExpression >::operator()
(
            const smtrat::expression::UnaryExpression & ec ) const  [inline]
```

15.225 std::hash< smtrat::lra::Variable< T1, T2 > > Struct Template Reference

Implements std::hash for sort value.

```
#include <Variable.h>
```

Public Member Functions

- size_t operator() (const smtrat::lra::Variable< T1, T2 > &_lraVar) const

15.225.1 Detailed Description

template<typename T1, typename T2>
struct std::hash< smtrat::lra::Variable< T1, T2 > >

Implements std::hash for sort value.

15.225.2 Member Function Documentation

15.225.2.1 operator()()

```
template<typename T1 , typename T2 >
size_t std::hash< smtrat::lra::Variable< T1, T2 > >::operator() (
            const smtrat::lra::Variable< T1, T2 > & _lraVar ) const  [inline]
```

Parameters

_lraVar	The LRA-variable to get the hash for.

Returns The hash of the given LRA-variable.

15.226 std::hash< smtrat::vs::Substitution > Struct Template Reference

```
#include <Substitution.h>
```

Public Member Functions

- size_t operator() (const smtrat::vs::Substitution &_substitution) const

15.226.1 Member Function Documentation

15.226.1.1 operator()()

```
size_t std::hash< smtrat::vs::Substitution >::operator() (
            const smtrat::vs::Substitution & _substitution ) const  [inline]
```

15.227 std::hash< std::vector< T > > Struct Template Reference

```
#include <ExpressionHash.h>
```

Public Member Functions

- std::size_t operator() (const std::vector< T > &v) const

15.227.1 Member Function Documentation

15.227.1.1 operator()()

```
template<typename T >
std::size_t std::hash< std::vector< T > >::operator() (
            const std::vector< T > & v ) const  [inline]
```

15.228 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

15.228.1 Constructor & Destructor Documentation

15.228.1.1 hash_combiner()

```
std::hash_combiner::hash_combiner (
            std::size_t _seed = 0 )  [inline]
```

15.228.2 Member Function Documentation

15.228.2.1 operator std::size_t()

```
std::hash_combiner::operator std::size_t ( ) const  [inline]
```

15.228.2.2 operator()() [1/2]

```
template<typename T >
hash_combiner& std::hash_combiner::operator() (
            const T & t )  [inline]
```

15.228.2.3 operator()() [2/2]

```
hash_combiner& std::hash_combiner::operator() (
            std::size_t h )  [inline]
```

15.228.3 Field Documentation

15.228.3.1 seed

```
std::size_t std::hash_combiner::seed
```

15.229 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

15.229.1 Constructor & Destructor Documentation

15.229.1.1 Heap()

```
template<class Comp>
Minisat::Heap< Comp >::Heap (
            const Comp & c ) [inline]
```

15.229.2 Member Function Documentation

15.229.2.1 build()

```
template<class Comp>
void Minisat::Heap< Comp >::build (
            vec< int > & ns ) [inline]
```

15.229.2.2 clear()

```
template<class Comp>
void Minisat::Heap< Comp >::clear (
            bool dealloc = false ) [inline]
```

15.229.2.3 decrease()

```
template<class Comp>
void Minisat::Heap< Comp >::decrease (
            int n ) [inline]
```

15.229.2.4 empty()

```
template<class Comp>
bool Minisat::Heap< Comp >::empty ( ) const [inline]
```

15.229.2.5 increase()

```
template<class Comp>
void Minisat::Heap< Comp >::increase (
            int n ) [inline]
```

15.229.2.6 inHeap()

```
template<class Comp>
bool Minisat::Heap< Comp >::inHeap (
            int n ) const [inline]
```

15.229.2.7 insert() template<class Comp>

```
void Minisat::Heap< Comp >::insert (
            int n )  [inline]
```

15.229.2.8 operator[]() template<class Comp>

```
int Minisat::Heap< Comp >::operator[] (
            int index ) const  [inline]
```

15.229.2.9 print() template<class Comp>

```
void Minisat::Heap< Comp >::print ( ) const  [inline]
```

15.229.2.10 removeMin() template<class Comp>

```
int Minisat::Heap< Comp >::removeMin ( )  [inline]
```

15.229.2.11 size() template<class Comp>

```
int Minisat::Heap< Comp >::size ( ) const  [inline]
```

15.229.2.12 update() template<class Comp>

```
void Minisat::Heap< Comp >::update (
            int n )  [inline]
```

15.230 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

15.230.1 Detailed Description

Parses hexadecimals: `#x[0-9a-fA-F]+`

15.230.2 Member Typedef Documentation

15.230.2.1 ITRange typedef boost::iterator_range<Iterator> smtrat::parser::HexadecimalParser::ITRange

15.230.3 Constructor & Destructor Documentation

15.230.3.1 HexadecimalParser() smtrat::parser::HexadecimalParser::HexadecimalParser ( ) [inline]

15.230.4 Member Function Documentation

15.230.4.1 build()

```
FixedWidthConstant<Integer> smtrat::parser::HexadecimalParser::build (
    const ITRange & itr,
    const Integer & val ) [inline]
```

15.230.5 Field Documentation

15.230.5.1 main

`qi::rule<Iterator, FixedWidthConstant<Integer>), Skipper, qi::locals<Integer> > smtrat::parser::HexadecimalParser::main`

15.230.5.2 main2

`qi::rule<Iterator, FixedWidthConstant<Integer>), Skipper> smtrat::parser↩ ::HexadecimalParser::main2`

15.230.5.3 number

`qi::uint_parser<Integer,16,1,-1> smtrat::parser::HexadecimalParser::number`

15.231 smtrat::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 ()

15.231.1 Constructor & Destructor Documentation

15.231.1.1 HistoryNode() [1/2] smtrat::icp::HistoryNode::HistoryNode ( ) [inline]

Methods.

15.231.1.2 HistoryNode() [2/2] smtrat::icp::HistoryNode::HistoryNode (const EvalDoubleIntervalMap & _intervals) [inline]

15.231.1.3 ~HistoryNode() smtrat::icp::HistoryNode::~HistoryNode ( ) [inline]

15.231.2 Member Function Documentation

15.231.2.1 addContraction() void smtrat::icp::HistoryNode::addContraction (ContractionCandidate * _candidate, const set_icpVariable & _variables) [inline]

15.231.2.2 addInfeasibleConstraint() bool smtrat::icp::HistoryNode::addInfeasibleConstraint (const ConstraintT & _constraint, bool _addOnlyConstraint = false) [inline]

15.231.2.3 addInfeasibleVariable() bool smtrat::icp::HistoryNode::addInfeasibleVariable (const IcpVariable * _variable, bool _addOnlyVariable = false) [inline]

15.231.2.4 addInterval() bool smtrat::icp::HistoryNode::addInterval (carl::Variable::Arg _var, const DoubleInterval & _interval) [inline]

updates or inserts an interval into the actual map

Parameters

_var	
_interval	

Returns true if only an update

15.231.2.5 addReason() void smtrat::icp::HistoryNode::addReason (carl::Variable::Arg _variable, const ConstraintT & _reason) [inline]

15.231.2.6 addReasons() [1/2]

```
void smtrat::icp::HistoryNode::addReasons (
    carl::Variable::Arg _variable,
    const FormulasT & _origins ) [inline]
```

15.231.2.7 addReasons() [2/2]

```
void smtrat::icp::HistoryNode::addReasons (
    carl::Variable::Arg _variable,
    const std::set< ConstraintT > & _reasons ) [inline]
```

15.231.2.8 addVariableReason()

```
void smtrat::icp::HistoryNode::addVariableReason (
    carl::Variable::Arg _variable,
    const IcpVariable * _reason ) [inline]
```

15.231.2.9 addVariableReasons()

```
void smtrat::icp::HistoryNode::addVariableReasons (
    carl::Variable::Arg _variable,
    set_icpVariable _variables ) [inline]
```

15.231.2.10 appliedConstraints() [1/2]

```
FormulaSetT smtrat::icp::HistoryNode::appliedConstraints
( ) [inline]
```

15.231.2.11 appliedConstraints() [2/2]

```
void smtrat::icp::HistoryNode::appliedConstraints (
    std::vector< FormulaT > & _result ) [inline]
```

15.231.2.12 appliedContractions()

```
std::set<const ContractionCandidate*> smtrat::icp::History↩
Node::appliedContractions ( ) [inline]
```

15.231.2.13 getCandidates()

```
const std::set<const ContractionCandidate*>& smtrat::icp::History↩
Node::getCandidates ( ) const [inline]
```

15.231.2.14 getInterval()

```
DoubleInterval& smtrat::icp::HistoryNode::getInterval (
    carl::Variable::Arg _variable ) [inline]
```

15.231.2.15 hasEmptyInterval()

```
bool smtrat::icp::HistoryNode::hasEmptyInterval ( ) const [inline]
```

15.231.2.16 intervals()

```
const EvalDoubleIntervalMap& smtrat::icp::HistoryNode::intervals ( )
const [inline]
```

Getters and Setters.

15.231.2.17 print()

```
void smtrat::icp::HistoryNode::print (
    std::ostream & _out = std::cout ) const [inline]
```

15.231.2.18 printReasons()

```
void smtrat::icp::HistoryNode::printReasons (
    std::ostream & _out = std::cout ) const [inline]
```

15.231.2.19 printVariableReasons()

```
void smtrat::icp::HistoryNode::printVariableReasons (
    std::ostream & _out = std::cout ) const [inline]
```

15.231.2.20 propagateStateInfeasibleConstraints()

```
void smtrat::icp::HistoryNode::propagateStateInfeasibleConstraints (
    HistoryNode * _target ) const [inline]
```

15.231.2.21 propagateStateInfeasibleVariables()

```
void smtrat::icp::HistoryNode::propagateStateInfeasibleVariables (
    HistoryNode * _target ) const [inline]
```

15.231.2.22 reasons() [1/2]

```
const std::map<carl::Variable, std::set<ConstraintT> >& smtrat::icp::HistoryNode::reasons ( ) const [inline]
```

15.231.2.23 reasons() [2/2]

```
std::set<ConstraintT>& smtrat::icp::HistoryNode::reasons (
    carl::Variable::Arg _variable ) [inline]
```

15.231.2.24 reset()

```
void smtrat::icp::HistoryNode::reset ( ) [inline]
```

15.231.2.25 rIntervals()

```
EvalDoubleIntervalMap& smtrat::icp::HistoryNode::rIntervals ( ) [inline]
```

15.231.2.26 rReasons()

```
std::map<carl::Variable, std::set<ConstraintT> >& smtrat::icp::HistoryNode::rReasons ( ) [inline]
```

15.231.2.27 rStateInfeasibleConstraints()

```
std::set<ConstraintT>& smtrat::icp::HistoryNode::rStateInfeasibleConstraints ( ) [inline]
```

15.231.2.28 rStateInfeasibleVariables()

```
set_icpVariable& smtrat::icp::HistoryNode::rStateInfeasibleVariables ( ) [inline]
```

15.231.2.29 rVariableReasons()

`std::map<carl::Variable, set_icpVariable>& smtrat::icp::HistoryNode::rVariableReasons ( ) [inline]`

15.231.2.30 setIntervals()

`void smtrat::icp::HistoryNode::setIntervals ( const EvalDoubleIntervalMap & _map ) [inline]`

15.231.2.31 stateInfeasibleConstraints()

`const std::set<ConstraintT>& smtrat::icp::HistoryNode::stateInfeasibleConstraints ( ) const [inline]`

15.231.2.32 stateInfeasibleVariables()

`set_icpVariable smtrat::icp::HistoryNode::stateInfeasibleVariables ( ) const [inline]`

15.231.2.33 variableHull()

`void smtrat::icp::HistoryNode::variableHull ( carl::Variable::Arg _variable, set_icpVariable & _result ) const [inline]`

15.231.2.34 variableReasons() [1/2]

`const std::map<carl::Variable, set_icpVariable>& smtrat::icp::HistoryNode::variableReasons ( ) [inline]`

15.231.2.35 variableReasons() [2/2]

`set_icpVariable smtrat::icp::HistoryNode::variableReasons ( carl::Variable::Arg _variable ) [inline]`

15.232 smtrat::ICEModule< Settings > Class Template Reference

`#include <ICEModule.h>`

Public Types

- typedef Settings SettingsType
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::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 ()
- 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 smtrat::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 _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.
- 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 &_branching↩Value) 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 _preferLeft↩Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩::vector< FormulaT >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩CaseWeak=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.

15.232.1 Member Typedef Documentation

15.232.1.1 SettingsType `template<typename Settings >`

```
typedef Settings smtrat::ICEModule< Settings >::SettingsType
```

15.232.2 Member Enumeration Documentation

15.232.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

15.232.3 Constructor & Destructor Documentation

15.232.3.1 ICEModule() `template<class Settings >`

```
smtrat::ICEModule< Settings >::ICEModule (
```

```
const ModuleInput * _formula,
Conditionals & _conditionals,
Manager * _manager = nullptr )
```

15.232.3.2 ~ICEModule() template<class Settings >

smtrat::ICEModule< Settings >::~ICEModule ( )

15.232.4 Member Function Documentation

15.232.4.1 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

_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.

15.232.4.2 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 >.

15.232.4.3 addCore() template<class 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

_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.

15.232.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.232.4.5 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.

15.232.4.6 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.

15.232.4.7 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.

15.232.4.8 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.

15.232.4.9 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.

15.232.4.10 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

_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.

15.232.4.11 allModels() `const std::vector<Model>& smtrat::Module::allModels ( ) const [inline], [inherited]`

Returns All satisfying assignments, if existent.

15.232.4.12 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.

15.232.4.13 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.

15.232.4.14 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.

15.232.4.15 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.

15.232.4.16 branchAt() [1/6]

```
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]
```

15.232.4.17 branchAt() [2/6]

```
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]
```

15.232.4.18 branchAt() [3/6]

```
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]
```

15.232.4.19 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.232.4.20 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            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]
```

15.232.4.21 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

15.232.4.22 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

15.232.4.23 checkCore() `template<class 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.

15.232.4.24 checkInfSubsetForMinimality()
```
void smtrat::Module::checkInfSubsetForMinimality (
    std::vector< FormulaSetT >::const_iterator _infsubset,
    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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.232.4.25 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.

15.232.4.26 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.232.4.27 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.232.4.28 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.232.4.29 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.232.4.30 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.232.4.31 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

15.232.4.32 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.

15.232.4.33 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.232.4.34 constraintsToInform() `const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.232.4.35 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

15.232.4.36 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)

15.232.4.37 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

_constraint	The constraint to remove from internal data structures.

15.232.4.38 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.232.4.39 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.232.4.40 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.232.4.41 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const  [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.232.4.42 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
            const std::vector< FormulaT > & _origins ) const  [protected], [inherited]
```

Parameters

_origins	

Returns

15.232.4.43 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.

15.232.4.44 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.

15.232.4.45 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

15.232.4.46 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

15.232.4.47 getBackendsAllModels()

`void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

15.232.4.48 getBackendsModel()

`void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

15.232.4.49 getInfeasibleSubsets() [1/2]

`void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

15.232.4.50 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.232.4.51 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.232.4.52 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.

15.232.4.53 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.232.4.54 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.232.4.55 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.232.4.56 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 reveived formula, if the Boolean is true.

Reimplemented from smtrat::Module.

15.232.4.57 hasLemmas() `bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

15.232.4.58 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).

15.232.4.59 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns A unique ID to identify this module instance.

15.232.4.60 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.

15.232.4.61 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.232.4.62 informBackends()

`void smtrat::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.

15.232.4.63 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, and smtrat::ICPModule< ICPSettings1 >.

15.232.4.64 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.

15.232.4.65 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::FPPModule< Settings >.

15.232.4.66 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.232.4.67 isLemmaLevel() `bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.232.4.68 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().

15.232.4.69 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.

15.232.4.70 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.232.4.71 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.232.4.72 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.

15.232.4.73 moduleName() `virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]`

Returns The name of the given module type as name.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::EQPreprocessingModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::VSModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::GBModule< Settings >, smtrat::IntEqModule< Setti smtrat::CubeLIAModule< Settings >, smtrat::CurryModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IncWidthModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::BEModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::SplitSOSModule< Settings >.

15.232.4.74 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

15.232.4.75 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

15.232.4.76 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula. TODO: disable this method

15.232.4.77 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula.

15.232.4.78 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to the backends of this module.

15.232.4.79 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to this module.

15.232.4.80 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.232.4.81 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

_out	The stream to print the assignment on.

15.232.4.82 printInfeasibleSubsets()

`void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.232.4.83 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

_out	The stream to print the assignment on.

15.232.4.84 printPassedFormula()

`void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.232.4.85 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.232.4.86 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.

15.232.4.87 receivedFormulaChecked()

```
void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]
```

Notifies that the received formulas has been checked.

15.232.4.88 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩InfeasibleSubset (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

15.232.4.89 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const [inline], [inherited]
```

15.232.4.90 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

_subformula	The sub formula of the received formula to remove.

Reimplemented from smtrat::Module.

15.232.4.91 removeCore() template<class 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

_subformula	The position of the subformula to remove.

Reimplemented from smtrat::Module.

15.232.4.92 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin
(
            ModuleInput::iterator _formula,
            const FormulaT & _origin )  [inline], [protected], [inherited]
```

15.232.4.93 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins )  [inline], [protected],
[inherited]
```

15.232.4.94 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const  [inline],
[inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.232.4.95 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.232.4.96 runBackends() [1/2]

```
virtual Answer smtrat::PModule::runBackends ( )  [inline],
[virtual], [inherited]
```

Reimplemented from smtrat::Module.

15.232.4.97 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.

15.232.4.98 setId()

```
void smtrat::Module::setId (
    std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.232.4.99 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.

15.232.4.100 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.

15.232.4.101 splitUnequalConstraint()

```
void smtrat::Module::splitUnequalConstraint (
    const FormulaT & _unequalConstraint ) [protected], [inherited]
```

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.232.4.102 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.232.4.103 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 >.

15.232.4.104 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.232.4.105 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.

15.232.4.106 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).

15.232.5 Field Documentation

15.232.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.232.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.232.5.3 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.

15.232.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.232.5.5 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

15.232.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.232.5.7 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.

15.232.5.8 mFirstUncheckedReceivedSubformula `ModuleInput::const_iterator smtrat::Module::m↩ FirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

15.232.5.9 mFoundAnswer `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.232.5.10 mFullCheck `bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.232.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.232.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.232.5.13 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.

15.232.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.232.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.232.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.232.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.232.5.18 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.

15.232.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplitting↩Variables [static], [inherited]`

Reusable splitting variables.

15.232.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.232.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.232.5.22 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.

15.232.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

15.232.5.24 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).

15.232.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.233 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 LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::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 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 _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

- 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 ()
- 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 &_branching↩Value) 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 _preferLeft↩Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩::vector< FormulaT >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩CaseWeak=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.

15.233.1 Member Typedef Documentation

15.233.1.1 SettingsType template<class Settings>

```
typedef Settings smtrat::ICPModule< Settings >::SettingsType
```

15.233.1.2 WeightMap template<class Settings>

```
typedef carl::FastPointerMap<Poly*, weights> smtrat::ICPModule< Settings >::WeightMap
```

15.233.2 Member Enumeration Documentation

15.233.2.1 LemmaType enum smtrat::Module::LemmaType : unsigned [strong], [inherited]

Enumerator

NORMAL	
PERMANENT	

15.233.3 Constructor & Destructor Documentation

15.233.3.1 ICPModule() template<class Settings >

```
smtrat::ICPModule< Settings >::ICPModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * const _manager = NULL )
```

Constructors:

15.233.3.2 ~ICPModule() template<class Settings >

```
smtrat::ICPModule< Settings >::~ICPModule ( )
```

Destructor:

15.233.4 Member Function Documentation

15.233.4.1 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.

15.233.4.2 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 >.

15.233.4.3 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

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.

15.233.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
            const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.233.4.5 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.

15.233.4.6 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.

15.233.4.7 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.

15.233.4.8 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.

15.233.4.9 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.

15.233.4.10 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

_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.

15.233.4.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const  [inline],
[inherited]
```

Returns All satisfying assignments, if existent.

15.233.4.12 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.

15.233.4.13 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.

15.233.4.14 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.

15.233.4.15 branchAt() [1/6]

```
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]
```

15.233.4.16 branchAt() [2/6]

```
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]
```

15.233.4.17 branchAt() [3/6]

```
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]
```

15.233.4.18 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.233.4.19 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            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]
```

15.233.4.20 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

15.233.4.21 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in smtrat::PModule.

15.233.4.22 checkCore()

```
template<class Settings >
Answer smtrat::ICPModule< Settings >::checkCore ( ) [virtual]
```

Checks the received formula for consistency.

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 SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

15.233.4.23 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
    std::vector< FormulaSetT >::const_iterator _infsubset,
    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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.233.4.24 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.

15.233.4.25 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.233.4.26 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.233.4.27 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.233.4.28 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.233.4.29 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.233.4.30 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

15.233.4.31 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

_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.

15.233.4.32 collectTheoryPropagations()

`void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.233.4.33 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.233.4.34 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

15.233.4.35 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)

15.233.4.36 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

_constraint	The constraint to remove from internal data structures.

15.233.4.37 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.233.4.38 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.233.4.39 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	SAT, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise).

Returns

Reimplemented from smtrat::Module.

15.233.4.40 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceived↩
VariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.233.4.41 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.233.4.42 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.

15.233.4.43 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.

15.233.4.44 freeSplittingVariable()

```
static void smtrat::Module::freeSplittingVariable (
    const FormulaT & _splittingVariable ) [inline], [static], [inherited]
```

15.233.4.45 generateTrivialInfeasibleSubset()

```
void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]
```

Stores the trivial infeasible subset being the set of received formulas.

15.233.4.46 getBackendsAllModels()

```
void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]
```

Stores all models of a backend in the list of all models of this module.

15.233.4.47 getBackendsModel()

```
void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]
```

15.233.4.48 getCurrentBoxAsFormulas() template<class Settings >

FormulasT smtrat::ICPModule< Settings >::getCurrentBoxAsFormulas ( ) const

15.233.4.49 getCurrentBoxAsIntervals() template<class Settings >

EvalRationalIntervalMap smtrat::ICPModule< Settings >::getCurrentBoxAsIntervals ( ) const

15.233.4.50 getInfeasibleSubsets() [1/2] void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]

Copies the infeasible subsets of the passed formula.

15.233.4.51 getInfeasibleSubsets() [2/2] std::vector< FormulaSetT > smtrat::Module::getInfeasible↩Subsets (

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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.233.4.52 getInformationRelevantFormulas() const std::set< FormulaT > & smtrat::Module::get↩InformationRelevantFormulas ( ) [protected], [inherited]

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.233.4.53 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.

15.233.4.54 getOrigins() [1/3] void smtrat::Module::getOrigins (

const FormulaT & *_formula*,
FormulaSetT & *_origins*) const [inline], [protected], [inherited]

Parameters

_formula	
_origins	

15.233.4.55 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.233.4.56 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.233.4.57 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 reveived formula, if the Boolean is true.

Reimplemented in smtrat::PModule.

15.233.4.58 hasLemmas()

```
bool smtrat::Module::hasLemmas ( ) [inline], [inherited]
```

Checks whether this module or any of its backends provides any lemmas.

15.233.4.59 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).

15.233.4.60 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns A unique ID to identify this module instance.

15.233.4.61 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.

15.233.4.62 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.233.4.63 informBackends() `void smtrat::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.

15.233.4.64 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

_constraint	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.

15.233.4.65 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.233.4.66 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::FPPModule< Settings >.

15.233.4.67 is_minimizing()

```
bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]
```

15.233.4.68 isLemmaLevel()

```
bool smtrat::Module::isLemmaLevel (
            LemmaLevel level ) [protected], [inherited]
```

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.233.4.69 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().

15.233.4.70 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.

15.233.4.71 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.233.4.72 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.233.4.73 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

_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.

15.233.4.74 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.

15.233.4.75 objective()

```
carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]
```

15.233.4.76 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
    const FormulaT & _origin ) const [protected], [inherited]
```

15.233.4.77 passedFormulaBegin()

```
ModuleInput::iterator smtrat::Module::passedFormulaBegin ( )
[inline], [protected], [inherited]
```

Returns An iterator to the end of the passed formula. TODO: disable this method

15.233.4.78 passedFormulaEnd()

```
ModuleInput::iterator smtrat::Module::passedFormulaEnd ( )
[inline], [protected], [inherited]
```

Returns An iterator to the end of the passed formula.

15.233.4.79 pPassedFormula()

```
const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline],
[inherited]
```

Returns A pointer to the formula passed to the backends of this module.

15.233.4.80 pReceivedFormula()

```
const ModuleInput* smtrat::Module::pReceivedFormula ( ) const
[inline], [inherited]
```

Returns A pointer to the formula passed to this module.

15.233.4.81 print()

```
void smtrat::Module::print (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.233.4.82 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

_out	The stream to print the assignment on.

15.233.4.83 printInfeasibleSubsets()

`void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.233.4.84 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

_out	The stream to print the assignment on.

15.233.4.85 printPassedFormula()

`void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.233.4.86 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.233.4.87 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

_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.

15.233.4.88 receivedFormulaChecked()

```
void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]
```

Notifies that the received formulas has been checked.

15.233.4.89 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩InfeasibleSubset (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

15.233.4.90 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const [inline], [inherited]
```

15.233.4.91 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.

15.233.4.92 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

formula	The sub formula of the received formula to remove.

Reimplemented from smtrat::Module.

15.233.4.93 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin
(
            ModuleInput::iterator _formula,
            const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.233.4.94 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

15.233.4.95 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline],
[inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.233.4.96 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.233.4.97 runBackends() [1/2]

`virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]`

Reimplemented in smtrat::PModule.

15.233.4.98 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

_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 in smtrat::PModule.

15.233.4.99 setId()

```
void smtrat::Module::setId (
    std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.233.4.100 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

_threadPriority	The priority to set this module's thread priority to.

15.233.4.101 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.

15.233.4.102 splitUnequalConstraint()

`void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.233.4.103 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.233.4.104 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 >.

15.233.4.105 updateLemmas()

`void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.233.4.106 updateModel()

`template<class Settings >`
`void smtrat::ICPModule< Settings >::updateModel ( ) const [virtual]`

Updates the model, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented from smtrat::Module.

15.233.4.107 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).

15.233.5 Field Documentation

15.233.5.1 mAllBackends

`std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.233.5.2 mAllModels

`std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.233.5.3 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.

15.233.5.4 mConstraintsToInform

`carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.233.5.5 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

15.233.5.6 mFirstPosInLastBranches

`std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.233.5.7 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.

15.233.5.8 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.

15.233.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.233.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.233.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.233.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.233.5.13 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.

15.233.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.233.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.233.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.233.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.233.5.18 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.

15.233.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]`

Reusable splitting variables.

15.233.5.20 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.233.5.21 mSmallerMusesCheckCounter

`unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.233.5.22 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.

15.233.5.23 mTheoryPropagations

`std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]`

15.233.5.24 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).

15.233.5.25 mVariableCounters

`std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.234 smtrat::icp::IcpVariable Class Reference

```
#include <IcpVariable.h>
```

Public Member Functions

- IcpVariable (carl::Variable::Arg _var, bool _original, ModuleInput::iterator _defaultPosition, EvalDouble↩ IntervalMap::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 isLinear ()
- 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)

15.234.1 Constructor & Destructor Documentation

15.234.1.1 IcpVariable()

```
smtrat::icp::IcpVariable::IcpVariable (
    carl::Variable::Arg _var,
    bool _original,
    ModuleInput::iterator _defaultPosition,
    EvalDoubleIntervalMap::iterator _intervalPos,
    const LRAVariable * _lraVar = NULL ) [inline]
```

15.234.1.2 ~IcpVariable()

```
smtrat::icp::IcpVariable::~IcpVariable ( ) [inline]
```

15.234.2 Member Function Documentation

15.234.2.1 addCandidate()

```
void smtrat::icp::IcpVariable::addCandidate (
    ContractionCandidate * _candidate ) [inline]
```

15.234.2.2 addCandidates()

```
void smtrat::icp::IcpVariable::addCandidates (
    const ContractionCandidates & _candidates ) [inline]
```

15.234.2.3 addOriginalConstraint()

```
void smtrat::icp::IcpVariable::addOriginalConstraint (
    const FormulaT & _constraint ) [inline]
```

15.234.2.4 candidates()

```
ContractionCandidates& smtrat::icp::IcpVariable::candidates ( ) [inline]
```

15.234.2.5 decrementActivity()

```
void smtrat::icp::IcpVariable::decrementActivity ( ) [inline]
```

15.234.2.6 externalLeftBound()

```
ModuleInput::iterator smtrat::icp::IcpVariable::externalLeft↩
Bound ( ) const [inline]
```

15.234.2.7 externalRightBound()

```
ModuleInput::iterator smtrat::icp::IcpVariable::externalRight↩
Bound ( ) const [inline]
```

15.234.2.8 incrementActivity()

```
void smtrat::icp::IcpVariable::incrementActivity ( ) [inline]
```

15.234.2.9 internalLeftBound()

```
const smtrat::FormulaT& smtrat::icp::IcpVariable::internalLeft↩
Bound ( ) const [inline]
```

15.234.2.10 internalRightBound()

```
const smtrat::FormulaT& smtrat::icp::IcpVariable::internal↩
RightBound ( ) const [inline]
```

15.234.2.11 interval()

```
const DoubleInterval& smtrat::icp::IcpVariable::interval ( ) const [inline]
```

15.234.2.12 intervalPosition()

```
EvalDoubleIntervalMap::const_iterator smtrat::icp::IcpVariable↩
::intervalPosition ( ) const [inline]
```

15.234.2.13 isActive()

```
bool smtrat::icp::IcpVariable::isActive ( ) const [inline]
```

15.234.2.14 isExternalUpdated()

`Updated smtrat::icp::IcpVariable::isExternalUpdated ( ) const [inline]`

15.234.2.15 isInternalBoundsSet()

`Updated smtrat::icp::IcpVariable::isInternalBoundsSet ( ) const [inline]`

15.234.2.16 isInternalUpdated()

`Updated smtrat::icp::IcpVariable::isInternalUpdated ( ) const [inline]`

15.234.2.17 isLinear()

`bool smtrat::icp::IcpVariable::isLinear ( ) [inline]`

15.234.2.18 isOriginal()

`bool smtrat::icp::IcpVariable::isOriginal ( ) const [inline]`

15.234.2.19 lraVar()

`const LRAVariable* smtrat::icp::IcpVariable::lraVar ( ) const [inline]`

15.234.2.20 operator<()

```
bool smtrat::icp::IcpVariable::operator< (
    IcpVariable const & rhs ) const [inline]
```

15.234.2.21 print()

```
void smtrat::icp::IcpVariable::print (
    std::ostream & _out = std::cout,
    bool _withContractionCandidates = false ) const [inline]
```

15.234.2.22 removeOriginalConstraint()

```
void smtrat::icp::IcpVariable::removeOriginalConstraint (
    const FormulaT & _constraint ) [inline]
```

15.234.2.23 setExternalLeftBound()

```
void smtrat::icp::IcpVariable::setExternalLeftBound (
    ModuleInput::iterator _left ) [inline]
```

15.234.2.24 setExternalModified()

`void smtrat::icp::IcpVariable::setExternalModified ( ) [inline]`

15.234.2.25 setExternalRightBound()

```
void smtrat::icp::IcpVariable::setExternalRightBound (
    ModuleInput::iterator _right ) [inline]
```

15.234.2.26 setExternalUnmodified() `void smtrat::icp::IcpVariable::setExternalUnmodified ( ) [inline]`

15.234.2.27 setInternalLeftBound() `void smtrat::icp::IcpVariable::setInternalLeftBound ( const smtrat::FormulaT & _left ) [inline]`

15.234.2.28 setInternalRightBound() `void smtrat::icp::IcpVariable::setInternalRightBound ( const smtrat::FormulaT & _right ) [inline]`

15.234.2.29 setInternalUnmodified() `void smtrat::icp::IcpVariable::setInternalUnmodified ( ) [inline]`

15.234.2.30 setInterval() `void smtrat::icp::IcpVariable::setInterval ( const DoubleInterval & _interval ) [inline]`

15.234.2.31 setLraVar() `void smtrat::icp::IcpVariable::setLraVar ( const LRAVariable * _lraVar ) [inline]`

15.234.2.32 setUnmodified() `void smtrat::icp::IcpVariable::setUnmodified ( ) [inline]`

15.234.2.33 var() `carl::Variable::Arg smtrat::icp::IcpVariable::var ( ) const [inline]`

15.234.3 Friends And Related Function Documentation

15.234.3.1 operator<< `std::ostream& operator<< ( std::ostream & os, const IcpVariable & _var ) [friend]`

15.235 smtrat::icp::icpVariableComp Struct Reference

`#include <IcpVariable.h>`

Public Member Functions

- bool operator() (const IcpVariable *const _lhs, const IcpVariable *const _rhs) const

15.235.1 Member Function Documentation

15.235.1.1 operator()() `bool smtrat::icp::icpVariableComp::operator() ( const IcpVariable *const _lhs, const IcpVariable *const _rhs ) const [inline]`

15.236 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

15.236.1 Constructor & Destructor Documentation

15.236.1.1 Identifier() [1/5] `smtrat::parser::Identifier::Identifier ( ) [inline]`

15.236.1.2 Identifier() [2/5] `smtrat::parser::Identifier::Identifier ( const std::string & symbol ) [inline]`

15.236.1.3 Identifier() [3/5] `smtrat::parser::Identifier::Identifier ( const std::string & symbol, const std::vector< std::size_t > & indices ) [inline]`

15.236.1.4 Identifier() [4/5] `smtrat::parser::Identifier::Identifier ( const std::string & symbol, const std::vector< Integer > & indices ) [inline]`

15.236.1.5 Identifier() [5/5] `smtrat::parser::Identifier::Identifier ( const Identifier & i ) [inline]`

15.236.1.6 ~Identifier() `smtrat::parser::Identifier::~Identifier ( ) [inline]`

15.236.2 Member Function Documentation

15.236.2.1 operator std::string() `smtrat::parser::Identifier::operator std::string ( ) const [inline]`

15.236.2.2 operator=() `Identifier& smtrat::parser::Identifier::operator= ( const Identifier & i ) [inline]`

15.236.3 Field Documentation

15.236.3.1 indices `std::vector<std::size_t>* smtrat::parser::Identifier::indices`

15.236.3.2 symbol `std::string smtrat::parser::Identifier::symbol`

15.237 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

15.237.1 Constructor & Destructor Documentation

15.237.1.1 IdentifierParser() `smtrat::parser::IdentifierParser::IdentifierParser ( ) [inline]`

15.237.2 Field Documentation

15.237.2.1 indexed `qi::rule<Iterator, Identifier(), Skipper> smtrat::parser::Identifier↩ Parser::indexed`

15.237.2.2 main `qi::rule<Iterator, Identifier(), Skipper> smtrat::parser::IdentifierParser↩ ::main`

15.237.2.3 numeral `NumeralParser smtrat::parser::IdentifierParser::numeral`

15.237.2.4 symbol `SymbolParser smtrat::parser::IdentifierParser::symbol`

15.238 smtrat::cad::debug::IDSanitizer Struct Reference

`#include <TikzHistoryPrinter.h>`

Public Member Functions

- IDSanitizer (const std::string &prefix)
- std::string operator() (const std::string &raw)

15.238.1 Constructor & Destructor Documentation

15.238.1.1 IDSanitizer() `smtrat::cad::debug::IDSanitizer::IDSanitizer ( const std::string & prefix ) [inline]`

15.238.2 Member Function Documentation

15.238.2.1 operator()() `std::string smtrat::cad::debug::IDSanitizer::operator() ( const std::string & raw ) [inline]`

15.239 smtrat::mcsat::fm::IgnoreCoreSettings Struct Reference

`#include <Explanation.h>`

Static Public Attributes

- static constexpr bool use_all_constraints = true

15.239.1 Field Documentation

15.239.1.1 use_all_constraints `constexpr bool smtrat::mcsat::fm::IgnoreCoreSettings::use_all_constraints = true [static], [constexpr]`

15.240 smtrat::IncWidthModule< Settings > Class Template Reference

`#include <IncWidthModule.h>`

Public Types

- typedef Settings SettingsType
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- IncWidthModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=NULL)

  Constructors.
- ~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 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 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 _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)

  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 &_branching↩Value) 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 _preferLeft↩Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩::vector< FormulaT >())

- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩CaseWeak=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.

15.240.1 Member Typedef Documentation

15.240.1.1 SettingsType

```
template<typename Settings >
typedef Settings smtrat::IncWidthModule< Settings >::SettingsType
```

15.240.2 Member Enumeration Documentation

15.240.2.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

15.240.3 Constructor & Destructor Documentation

15.240.3.1 IncWidthModule()

```
template<class Settings >
smtrat::IncWidthModule< Settings >::IncWidthModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = NULL )
```

Constructors.

15.240.3.2 ~IncWidthModule()

```
template<class Settings >
smtrat::IncWidthModule< Settings >::~IncWidthModule ( )
```

15.240.4 Member Function Documentation

15.240.4.1 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.

15.240.4.2 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 >.

15.240.4.3 addCore()

```
template<class 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

_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.

15.240.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.240.4.5 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.

15.240.4.6 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.

15.240.4.7 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.

15.240.4.8 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.

15.240.4.9 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.

15.240.4.10 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

_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.

15.240.4.11 allModels()

`const std::vector<Model>& smtrat::Module::allModels ( ) const [inline], [inherited]`

Returns All satisfying assignments, if existent.

15.240.4.12 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.

15.240.4.13 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.

15.240.4.14 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.

15.240.4.15 branchAt() [1/6]

```
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]
```

15.240.4.16 branchAt() [2/6]

```
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]
```

15.240.4.17 branchAt() [3/6]

```
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]
```

15.240.4.18 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.240.4.19 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P
>> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            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]
```

15.240.4.20 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false )  [inline], [protected], [inherited]
```

15.240.4.21 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in smtrat::PModule.

15.240.4.22 checkCore()

```
template<class Settings >
Answer smtrat::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 smtrat::Module.

15.240.4.23 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
            std::vector< FormulaSetT >::const_iterator _infsubset,
            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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.240.4.24 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.

15.240.4.25 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.240.4.26 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.240.4.27 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.240.4.28 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.240.4.29 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.240.4.30 collectOrigins() [1/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

15.240.4.31 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

_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.

15.240.4.32 collectTheoryPropagations()

`void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.240.4.33 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ToInform ( ) const [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.240.4.34 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

15.240.4.35 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)

15.240.4.36 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

_constraint	The constraint to remove from internal data structures.

15.240.4.37 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.240.4.38 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.240.4.39 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.240.4.40 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.240.4.41 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.240.4.42 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.

15.240.4.43 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.

15.240.4.44 freeSplittingVariable()

```
static void smtrat::Module::freeSplittingVariable (
    const FormulaT & _splittingVariable ) [inline], [static], [inherited]
```

15.240.4.45 generateTrivialInfeasibleSubset()

```
void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]
```

Stores the trivial infeasible subset being the set of received formulas.

15.240.4.46 getBackendsAllModels()

```
void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]
```

Stores all models of a backend in the list of all models of this module.

15.240.4.47 getBackendsModel()

```
void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]
```

15.240.4.48 getInfeasibleSubsets() [1/2]

```
void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]
```

Copies the infeasible subsets of the passed formula.

15.240.4.49 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.240.4.50 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.240.4.51 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.

15.240.4.52 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.240.4.53 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.240.4.54 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.240.4.55 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 reveived formula, if the Boolean is true.

Reimplemented in smtrat::PModule.

15.240.4.56 hasLemmas()

`bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

15.240.4.57 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).

15.240.4.58 id()

`std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns A unique ID to identify this module instance.

15.240.4.59 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.

15.240.4.60 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.240.4.61 informBackends()

```
void smtrat::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.

15.240.4.62 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, and smtrat::ICPModule< ICPSettings1 >.

15.240.4.63 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.240.4.64 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::FPPModule< Settings >.

15.240.4.65 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.240.4.66 isLemmaLevel() `bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.240.4.67 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().

15.240.4.68 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.

15.240.4.69 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.240.4.70 model()

```
const Model& smtrat::Module::model ( ) const [inline], [inherited]
```

Returns The assignment of the current satisfiable result, if existent.

15.240.4.71 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

_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.

15.240.4.72 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.

15.240.4.73 objective()

```
carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]
```

15.240.4.74 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
    const FormulaT & _origin ) const [protected], [inherited]
```

15.240.4.75 passedFormulaBegin() `ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula. TODO: disable this method

15.240.4.76 passedFormulaEnd() `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula.

15.240.4.77 pPassedFormula() `const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to the backends of this module.

15.240.4.78 pReceivedFormula() `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to this module.

15.240.4.79 print() `void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.240.4.80 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

_out	The stream to print the assignment on.

15.240.4.81 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const  [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.240.4.82 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

_out	The stream to print the assignment on.

15.240.4.83 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const  [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.240.4.84 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const  [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.240.4.85 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.

15.240.4.86 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

15.240.4.87 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩
InfeasibleSubset (
            ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

15.240.4.88 receivedVariable()

```
bool smtrat::Module::receivedVariable (
            carl::Variable::Arg _var ) const [inline], [inherited]
```

15.240.4.89 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

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

15.240.4.90 removeCore()

```
template<class Settings >
void smtrat::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

_subformula	The position of the subformula to remove.

Reimplemented from smtrat::Module.

15.240.4.91 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.240.4.92 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]
```

15.240.4.93 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.240.4.94 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.240.4.95 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]
```

Reimplemented in smtrat::PModule.

15.240.4.96 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

_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 in smtrat::PModule.

15.240.4.97 setId()

```
void smtrat::Module::setId ( std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.240.4.98 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.

15.240.4.99 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.

15.240.4.100 splitUnequalConstraint()

```
void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]
```

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.240.4.101 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.240.4.102 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 >.

15.240.4.103 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.240.4.104 updateModel() `template<class 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.

15.240.4.105 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).

15.240.5 Field Documentation

15.240.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.240.5.2 mAllModels

`std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.240.5.3 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.

15.240.5.4 mConstraintsToInform

`carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.240.5.5 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

15.240.5.6 mFirstPosInLastBranches

`std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.240.5.7 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.

15.240.5.8 mFirstUncheckedReceivedSubformula

`ModuleInput::const_iterator smtrat::Module::m↩FirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

15.240.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.240.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.240.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.240.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.240.5.13 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.

15.240.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.240.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.240.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.240.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.240.5.18 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.

15.240.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplitting↩Variables [static], [inherited]`

Reusable splitting variables.

15.240.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.240.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.240.5.22 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.

15.240.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

15.240.5.24 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).

15.240.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.241 smtrat::mcsat::onecellcad::firstlevel::indexed_root Struct Reference

`#include <first_level.h>`

Data Fields

- Poly poly
- size_t index

15.241.1 Field Documentation

15.241.1.1 index `size_t smtrat::mcsat::onecellcad::firstlevel::indexed_root::index`

15.241.1.2 poly `Poly smtrat::mcsat::onecellcad::firstlevel::indexed_root::poly`

15.242 smtrat::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

15.242.1 Constructor & Destructor Documentation

15.242.1.1 ~IndexedFunctionInstantiator()

```
virtual smtrat::parser::IndexedFunctionInstantiator↩
::~IndexedFunctionInstantiator ( ) [inline], [virtual]
```

15.242.2 Member Function Documentation

15.242.2.1 convert()

```
template<typename T >
bool smtrat::parser::IndexedFunctionInstantiator::convert (
            const std::vector< types::TermType > & from,
            std::vector< T > & to ) const [inline]
```

15.242.2.2 operator()()

```
virtual bool smtrat::parser::IndexedFunctionInstantiator::operator() (
            const std::vector< std::size_t > & ,
            const std::vector< types::TermType > & ,
            types::TermType & ,
            TheoryError & errors ) const [inline], [virtual]
```

15.243 smtrat::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.

15.243.1 Detailed Description

Represents the i-th root of a multivariate polynomial at its main variable (given an appropriate sample).

15.243.2 Constructor & Destructor Documentation

15.243.2.1 IndexedRoot() [1/2]

```
smtrat::cadcells::datastructures::IndexedRoot::IndexedRoot (
            PolyRef p,
            size_t i ) [inline]
```

15.243.2.2 IndexedRoot() [2/2]

```
smtrat::cadcells::datastructures::IndexedRoot::IndexedRoot ( )
[inline]
```

15.243.3 Field Documentation

15.243.3.1 index `size_t smtrat::cadcells::datastructures::IndexedRoot::index`

The index, must be > 0.

15.243.3.2 poly `PolyRef smtrat::cadcells::datastructures::IndexedRoot::poly`

A multivariate polynomial.

15.244 smtrat::cadcells::datastructures::IndexedRootOrdering Class Reference

Describes an ordering of IndexedRoots.

```
#include <roots.h>
```

Public Member Functions

- void add_leq (IndexedRoot first, IndexedRoot second)
- void add_eq (IndexedRoot first, IndexedRoot second)
- const auto & data () const
- const auto & leq () const
- const auto & geq () const
- bool leq_transitive (IndexedRoot first, IndexedRoot second) const
- std::optional< IndexedRoot > leq_transitive (IndexedRoot first, PolyRef poly) const
- std::optional< IndexedRoot > leq_transitive (PolyRef poly, IndexedRoot second) const
- boost::container::flat_set< PolyRef > polys () const

15.244.1 Detailed Description

Describes an ordering of IndexedRoots.

15.244.2 Member Function Documentation

15.244.2.1 add_eq()

```
void smtrat::cadcells::datastructures::IndexedRootOrdering::add_eq (
    IndexedRoot first,
    IndexedRoot second ) [inline]
```

15.244.2.2 add_leq()

```
void smtrat::cadcells::datastructures::IndexedRootOrdering::add_leq (
    IndexedRoot first,
    IndexedRoot second ) [inline]
```

15.244.2.3 data()

```
const auto& smtrat::cadcells::datastructures::IndexedRootOrdering::data ( )
const [inline]
```

15.244.2.4 geq()

```
const auto& smtrat::cadcells::datastructures::IndexedRootOrdering::geq ( )
const [inline]
```

15.244.2.5 leq()

```
const auto& smtrat::cadcells::datastructures::IndexedRootOrdering::leq ( )
const [inline]
```

15.244.2.6 leq_transitive() [1/3]

```
bool smtrat::cadcells::datastructures::IndexedRootOrdering↩
::leq_transitive (
            IndexedRoot first,
            IndexedRoot second ) const  [inline]
```

15.244.2.7 leq_transitive() [2/3]

```
std::optional<IndexedRoot> smtrat::cadcells::datastructures::↩
IndexedRootOrdering::leq_transitive (
            IndexedRoot first,
            PolyRef poly ) const  [inline]
```

15.244.2.8 leq_transitive() [3/3]

```
std::optional<IndexedRoot> smtrat::cadcells::datastructures::↩
IndexedRootOrdering::leq_transitive (
            PolyRef poly,
            IndexedRoot second ) const  [inline]
```

15.244.2.9 polys()

```
boost::container::flat_set<PolyRef> smtrat::cadcells::datastructures::↩
IndexedRootOrdering::polys ( ) const  [inline]
```

15.245 smtrat::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

15.245.1 Detailed Description

template<class Settings>
class smtrat::InequalitiesTable< Settings >

Datastructure for the GBModule.

A table of all inequalities and how they are reduced.

Author Sebastian Junges

15.245.2 Member Typedef Documentation

15.245.2.1 CellEntry `template<class Settings>`

```
typedef std::pair<unsigned, Polynomial> smtrat::InequalitiesTable< Settings >::CellEntry
```

15.245.2.2 Ideal `template<class Settings>`

```
typedef Settings::MultivariateIdeal smtrat::InequalitiesTable< Settings >::Ideal [protected]
```

15.245.2.3 Polynomial `template<class Settings>`

```
typedef Settings::PolynomialWithReasons smtrat::InequalitiesTable< Settings >::Polynomial
[protected]
```

15.245.2.4 RewriteRules `template<class Settings>`

```
typedef std::map<carl::Variable, std::pair<TermT, carl::BitVector> > smtrat::InequalitiesTable<
Settings >::RewriteRules
```

15.245.2.5 Row `template<class Settings>`

```
typedef std::pair<ModuleInput::const_iterator, RowEntry> smtrat::InequalitiesTable< Settings
>::Row
```

15.245.2.6 RowEntry `template<class Settings>`

```
typedef std::tuple<ModuleInput::iterator, carl::Relation, std::list<CellEntry> > smtrat::InequalitiesTable<
Settings >::RowEntry
```

15.245.2.7 Rows template<class Settings>

```
typedef std::map<ModuleInput::const_iterator, RowEntry, ModuleInput::IteratorCompare> smtrat::InequalitiesTabl
Settings >::Rows
```

15.245.3 Constructor & Destructor Documentation

15.245.3.1 InequalitiesTable() template<class Settings>

```
smtrat::InequalitiesTable< Settings >::InequalitiesTable (
            GBModule< Settings > * module )
```

15.245.4 Member Function Documentation

15.245.4.1 InsertReceivedFormula() template<class Settings>

```
Rows::iterator smtrat::InequalitiesTable< Settings >::InsertReceivedFormula (
            ModuleInput::const_iterator received )
```

15.245.4.2 popBacktrackPoint() template<class Settings>

```
void smtrat::InequalitiesTable< Settings >::popBacktrackPoint (
            unsigned nrBacktracks )
```

15.245.4.3 print() template<class Settings>

```
void smtrat::InequalitiesTable< Settings >::print (
            std::ostream & os = std::cout ) const
```

15.245.4.4 pushBacktrackPoint() template<class Settings>

```
void smtrat::InequalitiesTable< Settings >::pushBacktrackPoint ( )
```

15.245.4.5 reduceWRTGroebnerBasis() [1/3] template<class Settings>

```
Answer smtrat::InequalitiesTable< Settings >::reduceWRTGroebnerBasis (
            const Ideal & gb,
            const RewriteRules & rules )
```

15.245.4.6 reduceWRTGroebnerBasis() [2/3] template<class Settings>

```
Answer smtrat::InequalitiesTable< Settings >::reduceWRTGroebnerBasis (
            const std::list< typename Rows::iterator > & ineqToBeReduced,
            const Ideal & gb,
            const RewriteRules & rules )
```

15.245.4.7 reduceWRTGroebnerBasis() [3/3] template<class Settings>

```
bool smtrat::InequalitiesTable< Settings >::reduceWRTGroebnerBasis (
            typename Rows::iterator ,
            const Ideal & gb,
            const RewriteRules & rules )
```

15.245.4.8 reduceWRTVariableRewriteRules() [1/3] template<class Settings>

```
Answer smtrat::InequalitiesTable< Settings >::reduceWRTVariableRewriteRules (
            const RewriteRules & rules )
```

15.245.4.9 reduceWRTVariableRewriteRules() [2/3] template<class Settings>

```
Answer smtrat::InequalitiesTable< Settings >::reduceWRTVariableRewriteRules (
            const std::list< typename Rows::iterator > & ineqToBeReduced,
            const RewriteRules & rules )
```

15.245.4.10 reduceWRTVariableRewriteRules() [3/3] template<class Settings>

```
bool smtrat::InequalitiesTable< Settings >::reduceWRTVariableRewriteRules (
            typename Rows::iterator it,
            const RewriteRules & rules )
```

15.245.4.11 removeInequality() template<class Settings>

```
void smtrat::InequalitiesTable< Settings >::removeInequality (
            ModuleInput::const_iterator _formula )
```

15.245.5 Field Documentation

15.245.5.1 mBtnumber template<class Settings>

```
unsigned smtrat::InequalitiesTable< Settings >::mBtnumber [protected]
```

The actual number of backtrackpoints.

15.245.5.2 mLastRestart template<class Settings>

```
unsigned smtrat::InequalitiesTable< Settings >::mLastRestart [protected]
```

15.245.5.3 mModule template<class Settings>

```
GBModule<Settings>* smtrat::InequalitiesTable< Settings >::mModule [protected]
```

A pointer to the GBModule which uses this table.

15.245.5.4 mNewConstraints template<class Settings>

```
Rows::iterator smtrat::InequalitiesTable< Settings >::mNewConstraints [protected]
```

15.245.5.5 mReducedInequalities template<class Settings>

```
Rows smtrat::InequalitiesTable< Settings >::mReducedInequalities [protected]
```

A map of pointers from received iterators to rows.

15.246 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!=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

15.246.1 Detailed Description

template<typename T1, typename T2>
struct smtrat::lra::Bound< T1, T2 >::Info

Stores some additional information for a bound.

15.246.2 Constructor & Destructor Documentation

15.246.2.1 Info()

```
template<typename T1, typename T2>
smtrat::lra::Bound< T1, T2 >::Info::Info (
            ModuleInput::iterator _position )  [inline]
```

15.246.3 Field Documentation

15.246.3.1 complement

```
template<typename T1, typename T2>
const Bound<T1, T2>* smtrat::lra::Bound< T1, T2 >::Info::complement
```

15.246.3.2 exists

```
template<typename T1, typename T2>
bool smtrat::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.

15.246.3.3 neqRepresentation

```
template<typename T1, typename T2>
FormulaT smtrat::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!=0.

15.246.3.4 position

```
template<typename T1, typename T2>
ModuleInput::iterator smtrat::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.

15.246.3.5 updated

`template<typename T1, typename T2>`
`int smtrat::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).

15.247 smtrat::mcsat::InformationGetter Struct Reference

`#include <MCSATMixin.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

15.247.1 Field Documentation

15.247.1.1 abstractVariable

`std::function<Minisat::Var(const FormulaT&)> smtrat::mcsat::↩InformationGetter::abstractVariable`

15.247.1.2 getBoolVarValue

`std::function<Minisat::lbool(Minisat::Var)> smtrat::mcsat::↩InformationGetter::getBoolVarValue`

15.247.1.3 getClause

`std::function<const Minisat::Clause&(Minisat::CRef)> smtrat::mcsat::↩InformationGetter::getClause`

15.247.1.4 getClauses

`std::function<const Minisat::vec<Minisat::CRef>&()> smtrat::mcsat::↩InformationGetter::getClauses`

15.247.1.5 getDecisionLevel

`std::function<int(Minisat::Var)> smtrat::mcsat::Information↩Getter::getDecisionLevel`

15.247.1.6 getLearntClauses

`std::function<const Minisat::vec<Minisat::CRef>&()> smtrat↩::mcsat::InformationGetter::getLearntClauses`

15.247.1.7 getLitValue

`std::function<Minisat::lbool(Minisat::Lit)> smtrat::mcsat::InformationGetter::getLitValue`

15.247.1.8 getReason

`std::function<Minisat::CRef(Minisat::Var)> smtrat::mcsat::InformationGetter::getReason`

15.247.1.9 getTrailIndex

`std::function<int(Minisat::Var)> smtrat::mcsat::InformationGetter::getTrailIndex`

15.247.1.10 getVarValue

`std::function<Minisat::lbool(Minisat::Var)> smtrat::mcsat::InformationGetter::getVarValue`

15.247.1.11 getWatches

`std::function<const Minisat::vec<Minisat::Watcher>&(Minisat::Lit)> smtrat::mcsat::InformationGetter::getWatches`

15.247.1.12 isAbstractedFormula

`std::function<bool(const FormulaT&)> smtrat::mcsat::InformationGetter::isAbstractedFormula`

15.247.1.13 isTheoryAbstraction

`std::function<bool(Minisat::Var)> smtrat::mcsat::InformationGetter::isTheoryAbstraction`

15.247.1.14 newVar

`std::function<Minisat::Var()> smtrat::mcsat::InformationGetter::newVar`

15.247.1.15 reabstractLiteral

`std::function<const FormulaT&(Minisat::Lit)> smtrat::mcsat::InformationGetter::reabstractLiteral`

15.247.1.16 reabstractVariable

`std::function<const FormulaT&(Minisat::Var)> smtrat::mcsat::InformationGetter::reabstractVariable`

15.248 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

15.248.1 Member Function Documentation

15.248.1.1 instantiate()

```
template<typename V , typename T >
bool smtrat::parser::Instantiator< V, T >::instantiate (
            V v,
            const T & repl,
            types::TermType & subject )  [inline]
```

15.248.1.2 operator()() [1/6]

```
template<typename V , typename T >
bool smtrat::parser::Instantiator< V, T >::operator() (
            carl::Variable v )  [inline]
```

15.248.1.3 operator()() [2/6]

```
template<typename V , typename T >
bool smtrat::parser::Instantiator< V, T >::operator() (
            const FormulaT & f )  [inline]
```

15.248.1.4 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]
```

15.248.1.5 operator()() [4/6]

```
template<typename V , typename T >
template<typename Res >
bool smtrat::parser::Instantiator< V, T >::operator() (
            const Res &  )  [inline]
```

15.248.1.6 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]
```

15.248.1.7 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]
```

15.248.2 Field Documentation

15.248.2.1 replacement

```
template<typename V , typename T >
T smtrat::parser::Instantiator< V, T >::replacement  [protected]
```

15.248.2.2 result

```
template<typename V , typename T >
types::TermType smtrat::parser::Instantiator< V, T >::result  [protected]
```

15.248.2.3 var

```
template<typename V , typename T >
V smtrat::parser::Instantiator< V, T >::var  [protected]
```

15.249 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

15.249.1 Member Typedef Documentation

15.249.1.1 Value `typedef types::AttributeValue smtrat::parser::InstructionHandler::Value`

15.249.2 Constructor & Destructor Documentation

15.249.2.1 InstructionHandler() `smtrat::parser::InstructionHandler::InstructionHandler ( ) [inline]`

15.249.2.2 ~InstructionHandler() `virtual smtrat::parser::InstructionHandler::~InstructionHandler ( ) [inline], [virtual]`

15.249.3 Member Function Documentation

15.249.3.1 add() `virtual void smtrat::parser::InstructionHandler::add ( const FormulaT & f ) [pure virtual]`

Implemented in smtrat::Executor< Strategy >, and smtrat::parseformula::FormulaCollector.

15.249.3.2 addInstruction() `void smtrat::parser::InstructionHandler::addInstruction ( std::function< void() > bind ) [inline]`

15.249.3.3 addObjective()

```
virtual void smtrat::parser::InstructionHandler::addObjective (
    const Poly & p,
    OptimizationType ot )  [pure virtual]
```

Implemented in smtrat::Executor< Strategy >, and smtrat::parseformula::FormulaCollector.

15.249.3.4 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.

15.249.3.5 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.

15.249.3.6 check()

```
virtual void smtrat::parser::InstructionHandler::check ( )  [pure virtual]
```

Implemented in smtrat::Executor< Strategy >, and smtrat::parseformula::FormulaCollector.

15.249.3.7 cleanModel()

```
void smtrat::parser::InstructionHandler::cleanModel (
    smtrat::Model & model ) const  [inline]
```

15.249.3.8 declareFun()

```
virtual void smtrat::parser::InstructionHandler::declareFun (
    const carl::Variable &  )  [pure virtual]
```

Implemented in smtrat::Executor< Strategy >, and smtrat::parseformula::FormulaCollector.

15.249.3.9 declareSort()

```
virtual void smtrat::parser::InstructionHandler::declareSort (
    const std::string & ,
    const unsigned &  )  [pure virtual]
```

Implemented in smtrat::Executor< Strategy >, and smtrat::parseformula::FormulaCollector.

15.249.3.10 defineSort()

```
virtual void smtrat::parser::InstructionHandler::defineSort (
    const std::string & ,
    const std::vector< std::string > & ,
    const carl::Sort &  )  [pure virtual]
```

Implemented in smtrat::Executor< Strategy >, and smtrat::parseformula::FormulaCollector.

15.249.3.11 diagnostic() [1/2]

```
std::ostream& smtrat::parser::InstructionHandler::diagnostic ( ) [inline]
```

15.249.3.12 diagnostic() [2/2]

```
void smtrat::parser::InstructionHandler::diagnostic (
    const std::string & s ) [inline]
```

15.249.3.13 echo()

```
virtual void smtrat::parser::InstructionHandler::echo (
    const std::string & s ) [inline], [virtual]
```

15.249.3.14 eliminateQuantifiers()

```
virtual void smtrat::parser::InstructionHandler::eliminate↩
Quantifiers (
    const qe::QEQuery & q ) [pure virtual]
```

Implemented in smtrat::Executor< Strategy >, and smtrat::parseformula::FormulaCollector.

15.249.3.15 error()

```
OutputWrapper smtrat::parser::InstructionHandler::error ( ) [inline]
```

15.249.3.16 exit()

```
virtual void smtrat::parser::InstructionHandler::exit ( ) [pure virtual]
```

Implemented in smtrat::Executor< Strategy >, and smtrat::parseformula::FormulaCollector.

15.249.3.17 get_info()

```
const auto& smtrat::parser::InstructionHandler::get_info (
    const std::string & key ) const [inline]
```

15.249.3.18 getAllModels()

```
virtual void smtrat::parser::InstructionHandler::getAllModels ( ) [pure virtual]
```

Implemented in smtrat::Executor< Strategy >, and smtrat::parseformula::FormulaCollector.

15.249.3.19 getAssertions()

```
virtual void smtrat::parser::InstructionHandler::getAssertions ( ) [pure virtual]
```

Implemented in smtrat::Executor< Strategy >, and smtrat::parseformula::FormulaCollector.

15.249.3.20 getAssignment()

```
virtual void smtrat::parser::InstructionHandler::getAssignment ( ) [pure virtual]
```

Implemented in smtrat::Executor< Strategy >, and smtrat::parseformula::FormulaCollector.

15.249.3.21 getInfo()

```
void smtrat::parser::InstructionHandler::getInfo (
    const std::string & key ) [inline]
```

15.249.3.22 getModel()

```
virtual void smtrat::parser::InstructionHandler::getModel ( ) [pure virtual]
```

Implemented in smtrat::Executor< Strategy >, and smtrat::parseformula::FormulaCollector.

15.249.3.23 getObjectives()

```
virtual void smtrat::parser::InstructionHandler::getObjectives ( ) [pure virtual]
```

Implemented in smtrat::Executor< Strategy >, and smtrat::parseformula::FormulaCollector.

15.249.3.24 getOption()

```
void smtrat::parser::InstructionHandler::getOption (
    const std::string & key ) [inline]
```

15.249.3.25 getProof()

```
virtual void smtrat::parser::InstructionHandler::getProof ( ) [pure virtual]
```

Implemented in smtrat::Executor< Strategy >, and smtrat::parseformula::FormulaCollector.

15.249.3.26 getUnsatCore()

```
virtual void smtrat::parser::InstructionHandler::getUnsatCore ( ) [pure virtual]
```

Implemented in smtrat::Executor< Strategy >, and smtrat::parseformula::FormulaCollector.

15.249.3.27 getValue()

```
virtual void smtrat::parser::InstructionHandler::getValue (
    const std::vector< carl::Variable > & ) [pure virtual]
```

Implemented in smtrat::Executor< Strategy >, and smtrat::parseformula::FormulaCollector.

15.249.3.28 has_info()

```
bool smtrat::parser::InstructionHandler::has_info (
    const std::string & key ) const [inline]
```

15.249.3.29 hasInstructions()

```
bool smtrat::parser::InstructionHandler::hasInstructions ( ) const [inline]
```

15.249.3.30 info()

```
OutputWrapper smtrat::parser::InstructionHandler::info ( ) [inline]
```

15.249.3.31 option()

```
template<typename T >
T smtrat::parser::InstructionHandler::option (
            const std::string & key ) const  [inline]
```

15.249.3.32 pop()

```
virtual void smtrat::parser::InstructionHandler::pop (
            std::size_t  )  [pure virtual]
```

Implemented in smtrat::parseformula::FormulaCollector, and smtrat::Executor< Strategy >.

15.249.3.33 printInstruction()

```
bool smtrat::parser::InstructionHandler::printInstruction ( )
const  [inline]
```

15.249.3.34 push()

```
virtual void smtrat::parser::InstructionHandler::push (
            std::size_t  )  [pure virtual]
```

Implemented in smtrat::parseformula::FormulaCollector, and smtrat::Executor< Strategy >.

15.249.3.35 regular() [1/2]

```
std::ostream& smtrat::parser::InstructionHandler::regular ( )  [inline]
```

15.249.3.36 regular() [2/2]

```
void smtrat::parser::InstructionHandler::regular (
            const std::string & s )  [inline]
```

15.249.3.37 reset()

```
virtual void smtrat::parser::InstructionHandler::reset ( )  [inline], [virtual]
```

Reimplemented in smtrat::Executor< Strategy >, and smtrat::parseformula::FormulaCollector.

15.249.3.38 resetAssertions()

```
virtual void smtrat::parser::InstructionHandler::resetAssertions
( )  [pure virtual]
```

Implemented in smtrat::Executor< Strategy >, and smtrat::parseformula::FormulaCollector.

15.249.3.39 runInstructions()

```
void smtrat::parser::InstructionHandler::runInstructions ( )  [inline]
```

15.249.3.40 setArtificialVariables()

```
void smtrat::parser::InstructionHandler::setArtificial↩
Variables (
            std::vector< smtrat::ModelVariable > && vars )  [inline]
```

15.249.3.41 setInfo()

```
void smtrat::parser::InstructionHandler::setInfo (
    const Attribute & attr ) [inline]
```

15.249.3.42 setLogic()

```
virtual void smtrat::parser::InstructionHandler::setLogic (
    const carl::Logic & ) [pure virtual]
```

Implemented in smtrat::Executor< Strategy >, and smtrat::parseformula::FormulaCollector.

15.249.3.43 setOption()

```
void smtrat::parser::InstructionHandler::setOption (
    const Attribute & option ) [inline]
```

15.249.3.44 setStream()

```
void smtrat::parser::InstructionHandler::setStream (
    const std::string & s,
    std::ostream & os ) [inline], [protected]
```

15.249.3.45 warn()

```
OutputWrapper smtrat::parser::InstructionHandler::warn ( ) [inline]
```

15.249.4 Field Documentation

15.249.4.1 infos

```
VariantMap<std::string, Value> smtrat::parser::InstructionHandler::infos
[protected]
```

15.249.4.2 mDiagnostic

```
std::ostream smtrat::parser::InstructionHandler::mDiagnostic [protected]
```

15.249.4.3 mRegular

```
std::ostream smtrat::parser::InstructionHandler::mRegular [protected]
```

15.249.4.4 options

```
VariantMap<std::string, Value> smtrat::parser::InstructionHandler::options
[protected]
```

15.249.4.5 streams

```
std::map<std::string, std::ofstream> smtrat::parser::InstructionHandler↩
::streams [protected]
```

15.250 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

15.250.1 Constructor & Destructor Documentation

15.250.1.1 Int64Range()

```
Minisat::Int64Range::Int64Range (
    int64_t b,
    int64_t e ) [inline]
```

15.250.2 Field Documentation

15.250.2.1 begin

```
int64_t Minisat::Int64Range::begin
```

15.250.2.2 end

```
int64_t Minisat::Int64Range::end
```

15.251 smtrat::IntBlastModule< Settings > Class Template Reference

```
#include <IntBlastModule.h>
```

Public Types

- typedef Settings SettingsType
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- IntBlastModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=NULL)

  Constructors.
- ~IntBlastModule ()

  Destructor.
- 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 hasValidInfeasibleSubset () 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 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 &_branching↩ Value) 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 >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, 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.

15.251.1 Member Typedef Documentation

15.251.1.1 SettingsType `template<class Settings>`

```
typedef Settings smtrat::IntBlastModule< Settings >::SettingsType
```

15.251.2 Member Enumeration Documentation

15.251.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

15.251.3 Constructor & Destructor Documentation

15.251.3.1 IntBlastModule() `template<class Settings >`

```
smtrat::IntBlastModule< Settings >::IntBlastModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = NULL )
```

Constructors.

15.251.3.2 ~IntBlastModule() `template<class Settings >`

```
smtrat::IntBlastModule< Settings >::~IntBlastModule ( )
```

Destructor.

15.251.4 Member Function Documentation

15.251.4.1 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.

15.251.4.2 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 >.

15.251.4.3 addCore()

```
template<class Settings >
bool smtrat::IntBlastModule< 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.

15.251.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.251.4.5 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.

15.251.4.6 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.

15.251.4.7 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.

15.251.4.8 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.

15.251.4.9 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.

15.251.4.10 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

_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.

15.251.4.11 allModels() `const std::vector<Model>& smtrat::Module::allModels ( ) const [inline], [inherited]`

Returns All satisfying assignments, if existent.

15.251.4.12 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.

15.251.4.13 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.

15.251.4.14 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.

15.251.4.15 branchAt() [1/6]

```
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]
```

15.251.4.16 branchAt() [2/6]

```
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]
```

15.251.4.17 branchAt() [3/6]

```
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]
```

15.251.4.18 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.251.4.19 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
    const typename P::PolyType & _poly,
    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]
```

15.251.4.20 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false )  [inline], [protected], [inherited]
```

15.251.4.21 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in smtrat::PModule.

15.251.4.22 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.

15.251.4.23 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
            std::vector< FormulaSetT >::const_iterator _infsubset,
            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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.251.4.24 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.

15.251.4.25 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.251.4.26 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.251.4.27 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.251.4.28 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.251.4.29 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.251.4.30 collectOrigins() [1/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

15.251.4.31 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

_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.

15.251.4.32 collectTheoryPropagations()

`void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.251.4.33 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ToInform ( ) const [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.251.4.34 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

15.251.4.35 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)

15.251.4.36 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

_constraint	The constraint to remove from internal data structures.

15.251.4.37 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.251.4.38 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.251.4.39 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.251.4.40 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceived↩
VariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.251.4.41 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.251.4.42 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.

15.251.4.43 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.

15.251.4.44 freeSplittingVariable()

```
static void smtrat::Module::freeSplittingVariable (
    const FormulaT & _splittingVariable ) [inline], [static], [inherited]
```

15.251.4.45 generateTrivialInfeasibleSubset()

```
void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]
```

Stores the trivial infeasible subset being the set of received formulas.

15.251.4.46 getBackendsAllModels()

```
void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]
```

Stores all models of a backend in the list of all models of this module.

15.251.4.47 getBackendsModel()

```
void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]
```

15.251.4.48 getInfeasibleSubsets() [1/2]

```
void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]
```

Copies the infeasible subsets of the passed formula.

15.251.4.49 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.251.4.50 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.251.4.51 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.

15.251.4.52 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.251.4.53 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.251.4.54 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.251.4.55 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 reveived formula, if the Boolean is true.

Reimplemented in smtrat::PModule.

15.251.4.56 hasLemmas() bool smtrat::Module::hasLemmas ( ) [inline], [inherited]

Checks whether this module or any of its backends provides any lemmas.

15.251.4.57 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).

15.251.4.58 id() std::size_t smtrat::Module::id ( ) const [inline], [inherited]

Returns A unique ID to identify this module instance.

15.251.4.59 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.

15.251.4.60 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.251.4.61 informBackends()

```
void smtrat::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.

15.251.4.62 informCore()

```
template<class Settings >
bool smtrat::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

_constraint	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.

15.251.4.63 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.251.4.64 init()

```
template<class Settings >
void smtrat::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 smtrat::Module.

15.251.4.65 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.251.4.66 isLemmaLevel() `bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.251.4.67 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().

15.251.4.68 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.

15.251.4.69 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.251.4.70 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.251.4.71 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

_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.

15.251.4.72 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.

15.251.4.73 objective()

```
carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]
```

15.251.4.74 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
    const FormulaT & _origin ) const [protected], [inherited]
```

15.251.4.75 passedFormulaBegin()

```
ModuleInput::iterator smtrat::Module::passedFormulaBegin ( )
[inline], [protected], [inherited]
```

Returns An iterator to the end of the passed formula. TODO: disable this method

15.251.4.76 passedFormulaEnd()

```
ModuleInput::iterator smtrat::Module::passedFormulaEnd ( )
[inline], [protected], [inherited]
```

Returns An iterator to the end of the passed formula.

15.251.4.77 pPassedFormula()

```
const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline],
[inherited]
```

Returns A pointer to the formula passed to the backends of this module.

15.251.4.78 pReceivedFormula()

```
const ModuleInput* smtrat::Module::pReceivedFormula ( ) const
[inline], [inherited]
```

Returns A pointer to the formula passed to this module.

15.251.4.79 print()

```
void smtrat::Module::print (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.251.4.80 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

_out	The stream to print the assignment on.

15.251.4.81 printInfeasibleSubsets()

`void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.251.4.82 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

_out	The stream to print the assignment on.

15.251.4.83 printPassedFormula()

`void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.251.4.84 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.251.4.85 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.

15.251.4.86 receivedFormulaChecked()

```
void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]
```

Notifies that the received formulas has been checked.

15.251.4.87 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩InfeasibleSubset (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

15.251.4.88 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const [inline], [inherited]
```

15.251.4.89 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

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

15.251.4.90 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

_subformula	The position of the subformula to remove.

Reimplemented from smtrat::Module.

15.251.4.91 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin
(
            ModuleInput::iterator _formula,
            const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.251.4.92 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

15.251.4.93 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline],
[inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.251.4.94 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.251.4.95 runBackends() [1/2]

`virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]`

Reimplemented in smtrat::PModule.

15.251.4.96 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

_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 in smtrat::PModule.

15.251.4.97 setId()

```
void smtrat::Module::setId (
    std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.251.4.98 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.

15.251.4.99 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.

15.251.4.100 splitUnequalConstraint()

`void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.251.4.101 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.251.4.102 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 >.

15.251.4.103 updateLemmas()

`void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.251.4.104 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.

15.251.4.105 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).

15.251.5 Field Documentation

15.251.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.251.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.251.5.3 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.

15.251.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.251.5.5 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

15.251.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.251.5.7 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.

15.251.5.8 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.

15.251.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.251.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.251.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.251.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.251.5.13 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.

15.251.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.251.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.251.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.251.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.251.5.18 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.

15.251.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]`

Reusable splitting variables.

15.251.5.20 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.251.5.21 mSmallerMusesCheckCounter

`unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.251.5.22 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.

15.251.5.23 mTheoryPropagations

`std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]`

15.251.5.24 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).

15.251.5.25 mVariableCounters

`std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.252 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 LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- IntEqModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=NULL)

  Constructors.
- ~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 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 _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)

  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 &_branching↩Value) 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 _preferLeft↩Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩::vector< FormulaT >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩CaseWeak=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.

15.252.1 Detailed Description

template<typename Settings>
class smtrat::IntEqModule< Settings >

A module which checks whether the equations contained in the received (linear integer) formula have a solution.

15.252.2 Member Typedef Documentation

15.252.2.1 SettingsType

```
template<typename Settings >
typedef Settings smtrat::IntEqModule< Settings >::SettingsType
```

15.252.3 Member Enumeration Documentation

15.252.3.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

15.252.4 Constructor & Destructor Documentation

15.252.4.1 IntEqModule()

```
template<class Settings >
smtrat::IntEqModule< Settings >::IntEqModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = NULL )
```

Constructors.

15.252.4.2 ~IntEqModule()

```
template<typename Settings >
smtrat::IntEqModule< Settings >::~IntEqModule ( ) [inline]
```

15.252.5 Member Function Documentation

15.252.5.1 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.

15.252.5.2 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

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

15.252.5.3 addCore()

```
template<class 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

_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.

15.252.5.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.252.5.5 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.

15.252.5.6 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.

15.252.5.7 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.

15.252.5.8 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.

15.252.5.9 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

_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.

15.252.5.10 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

_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.

15.252.5.11 allModels()

`const std::vector<Model>& smtrat::Module::allModels ( ) const [inline], [inherited]`

Returns All satisfying assignments, if existent.

15.252.5.12 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.

15.252.5.13 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.

15.252.5.14 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.

15.252.5.15 branchAt() [1/6]

```
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]
```

15.252.5.16 branchAt() [2/6]

```
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]
```

15.252.5.17 branchAt() [3/6]

```
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]
```

15.252.5.18 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.252.5.19 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P
>> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            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]
```

15.252.5.20 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false )  [inline], [protected], [inherited]
```

15.252.5.21 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in smtrat::PModule.

15.252.5.22 checkCore()

```
template<class 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 smtrat::Module.

15.252.5.23 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
            std::vector< FormulaSetT >::const_iterator _infsubset,
            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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.252.5.24 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.

15.252.5.25 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.252.5.26 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.252.5.27 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.252.5.28 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.252.5.29 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.252.5.30 collectOrigins() [1/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

15.252.5.31 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

_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.

15.252.5.32 collectTheoryPropagations()

`void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.252.5.33 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ToInform ( ) const [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.252.5.34 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::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.252.5.35 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)

15.252.5.36 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

_constraint	The constraint to remove from internal data structures.

15.252.5.37 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.252.5.38 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.252.5.39 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.252.5.40 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.252.5.41 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.252.5.42 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.

15.252.5.43 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.

15.252.5.44 freeSplittingVariable()

```
static void smtrat::Module::freeSplittingVariable (
    const FormulaT & _splittingVariable ) [inline], [static], [inherited]
```

15.252.5.45 generateTrivialInfeasibleSubset()

```
void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]
```

Stores the trivial infeasible subset being the set of received formulas.

15.252.5.46 getBackendsAllModels()

```
void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]
```

Stores all models of a backend in the list of all models of this module.

15.252.5.47 getBackendsModel()

```
void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]
```

15.252.5.48 getInfeasibleSubsets() [1/2]

```
void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]
```

Copies the infeasible subsets of the passed formula.

15.252.5.49 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.252.5.50 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.252.5.51 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.

15.252.5.52 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.252.5.53 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.252.5.54 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.252.5.55 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 reveived formula, if the Boolean is true.

Reimplemented in smtrat::PModule.

15.252.5.56 hasLemmas()

`bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

15.252.5.57 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).

15.252.5.58 id()

`std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns A unique ID to identify this module instance.

15.252.5.59 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.

15.252.5.60 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.252.5.61 informBackends()

```
void smtrat::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.

15.252.5.62 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, and smtrat::ICPModule< ICPSettings1 >.

15.252.5.63 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.252.5.64 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::FPPModule< Settings >.

15.252.5.65 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.252.5.66 isLemmaLevel() `bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.252.5.67 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().

15.252.5.68 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.

15.252.5.69 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.252.5.70 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.252.5.71 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

_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.

15.252.5.72 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.

15.252.5.73 objective() `carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

15.252.5.74 originInReceivedFormula() `bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

15.252.5.75 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula. TODO: disable this method

15.252.5.76 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula.

15.252.5.77 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to the backends of this module.

15.252.5.78 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to this module.

15.252.5.79 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.252.5.80 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

_out	The stream to print the assignment on.

15.252.5.81 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const  [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.252.5.82 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

_out	The stream to print the assignment on.

15.252.5.83 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const  [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.252.5.84 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const  [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.252.5.85 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.

15.252.5.86 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

15.252.5.87 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

15.252.5.88 receivedVariable()

`bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

15.252.5.89 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

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

15.252.5.90 removeCore()

`template<class 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

_subformula	The position of the subformula to remove.

Reimplemented from smtrat::Module.

15.252.5.91 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.252.5.92 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]
```

15.252.5.93 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.252.5.94 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.252.5.95 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]
```

Reimplemented in smtrat::PModule.

15.252.5.96 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

_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 in smtrat::PModule.

15.252.5.97 setId()

```
void smtrat::Module::setId ( std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.252.5.98 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.

15.252.5.99 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.

15.252.5.100 splitUnequalConstraint()

```
void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]
```

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.252.5.101 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.252.5.102 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 >.

15.252.5.103 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.252.5.104 updateModel() `template<class 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.

15.252.5.105 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).

15.252.6 Field Documentation

15.252.6.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.252.6.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.252.6.3 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.

15.252.6.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.252.6.5 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

15.252.6.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.252.6.7 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.

15.252.6.8 mFirstUncheckedReceivedSubformula `ModuleInput::const_iterator smtrat::Module::m↩FirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

15.252.6.9 mFoundAnswer `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.252.6.10 mFullCheck `bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.252.6.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.252.6.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.252.6.13 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.

15.252.6.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.252.6.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.252.6.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.252.6.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.252.6.18 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.

15.252.6.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplitting↩Variables [static], [inherited]`

Reusable splitting variables.

15.252.6.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.252.6.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.252.6.22 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.

15.252.6.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

15.252.6.24 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).

15.252.6.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.253 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 ()

15.253.1 Constructor & Destructor Documentation

15.253.1.1 IntervalPropagation()

```
smtrat::mcsat::icp::IntervalPropagation::IntervalPropagation (
    const std::vector< carl::Variable > & vars,
    const std::vector< FormulaT > & constraints,
    const Model & model ) [inline]
```

15.253.2 Member Function Documentation

15.253.2.1 execute() `std::optional<FormulaT> smtrat::mcsat::icp::IntervalPropagation::execute ( ) [inline]`

15.254 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

15.254.1 Constructor & Destructor Documentation

15.254.1.1 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]
```

15.254.2 Member Function Documentation

15.254.2.1 getHelpPrefixString()

```
static const char*& Minisat::Option::getHelpPrefixString ( )
[inline], [static], [protected], [inherited]
```

15.254.2.2 getOptionList()

```
static vec<Option*>& Minisat::Option::getOptionList ( )  [inline],
[static], [protected], [inherited]
```

15.254.2.3 getUsageString()

```
static const char*& Minisat::Option::getUsageString ( )  [inline],
[static], [protected], [inherited]
```

15.254.2.4 help()

```
virtual void Minisat::IntOption::help (
    bool verbose = false )  [inline], [virtual]
```

Implements Minisat::Option.

15.254.2.5 operator int32_t()

```
Minisat::IntOption::operator int32_t (
    void ) const [inline]
```

15.254.2.6 operator=()

```
IntOption& Minisat::IntOption::operator= (
    int32_t x ) [inline]
```

15.254.3 Field Documentation

15.254.3.1 category

```
const char* Minisat::Option::category [protected], [inherited]
```

15.254.3.2 description

```
const char* Minisat::Option::description [protected], [inherited]
```

15.254.3.3 name

```
const char* Minisat::Option::name [protected], [inherited]
```

15.254.3.4 range

```
IntRange Minisat::IntOption::range [protected]
```

15.254.3.5 type_name

```
const char* Minisat::Option::type_name [protected], [inherited]
```

15.254.3.6 value

```
int32_t Minisat::IntOption::value [protected]
```

15.255 Minisat::IntRange Struct Reference

```
#include <Options.h>
```

Public Member Functions

- IntRange (int b, int e)

Data Fields

- int begin
- int end

15.255.1 Constructor & Destructor Documentation

15.255.1.1 IntRange()

```
Minisat::IntRange::IntRange (
    int b,
    int e ) [inline]
```

15.255.2 Field Documentation

15.255.2.1 begin

```
int Minisat::IntRange::begin
```

15.255.2.2 end

```
int Minisat::IntRange::end
```

15.256 smtrat::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)

15.256.1 Member Function Documentation

15.256.1.1 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]
```

15.257 smtrat::is_sample_outside< IsSampleOutsideAlgorithm::DEFAULT > 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)

15.257.1 Member Function Documentation

15.257.1.1 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]
```

15.258 smtrat::is_variant< T > Struct Template Reference

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

```
#include <VariantMap.h>
```

15.258.1 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.

15.259 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

15.259.1 Constructor & Destructor Documentation

15.259.1.1 ITEExpression()

```
smtrat::expression::ITEExpression::ITEExpression (
    ITEType ,
    Expression && _condition,
    Expression && _then,
    Expression && _else )  [inline]
```

15.259.2 Field Documentation

15.259.2.1 condition

```
Expression smtrat::expression::ITEExpression::condition
```

15.259.2.2 elsecase

```
Expression smtrat::expression::ITEExpression::elsecase
```

15.259.2.3 thencase

```
Expression smtrat::expression::ITEExpression::thencase
```

15.259.2.4 type

```
ITEType smtrat::expression::ITEExpression::type
```

15.260 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.

15.260.1 Constructor & Destructor Documentation

15.260.1.1 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.

15.260.2 Member Function Documentation

15.260.2.1 operator"!=() template<typename T >

```
bool benchmax::RandomizationAdaptor< T >::iterator::operator!= (
            const iterator & rhs ) const  [inline]
```

Compare two iterators.

15.260.2.2 operator*() template<typename T >

```
const T& benchmax::RandomizationAdaptor< T >::iterator::operator* ( )  [inline]
```

Dereference iterator.

15.260.2.3 operator++() template<typename T >

```
iterator& benchmax::RandomizationAdaptor< T >::iterator::operator++ ( )  [inline]
```

Increment iterator.

15.261 smtrat::ModuleInput::IteratorCompare Struct Reference

```
#include <ModuleInput.h>
```

Public Member Functions

- bool operator() (const_iterator i1, const_iterator i2) const

15.261.1 Member Function Documentation

15.261.1.1 operator()() bool smtrat::ModuleInput::IteratorCompare::operator() (

```
            const_iterator i1,
            const_iterator i2 ) const  [inline]
```

15.262 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.

15.262.1 Detailed Description Represents a set of jobs, constructed as the cartesian product of a set of tools and a set of benchmarks.

15.262.2 Constructor & Destructor Documentation

15.262.2.1 Jobs()

```
benchmax::Jobs::Jobs (
    const Tools & tools,
    const BenchmarkSet & benchmarks ) [inline]
```

Construct jobs from a set of tools and a set of benchmark files.

15.262.3 Member Function Documentation

15.262.3.1 begin()

```
auto benchmax::Jobs::begin ( ) const [inline]
```

Begin iterator.

15.262.3.2 end()

```
auto benchmax::Jobs::end ( ) const [inline]
```

End iterator.

15.262.3.3 files()

```
const auto& benchmax::Jobs::files ( ) const [inline]
```

Returns the set of files.

15.262.3.4 randomized()

```
auto benchmax::Jobs::randomized ( ) const [inline]
```

Returns all jobs in a pseudo-randomized order.

15.262.3.5 size()

```
auto benchmax::Jobs::size ( ) const [inline]
```

Returns the overall number of jobs.

15.262.3.6 tools()

```
const auto& benchmax::Jobs::tools ( ) const [inline]
```

Returns the set of tools.

15.263 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)

15.263.1 Member Function Documentation

15.263.1.1 rewrite_and()

```
FormulaT smtrat::JunctorMerger::rewrite_and (
    const FormulaT & formula,
    FormulaSetT && subformulas ) [inline]
```

15.263.1.2 rewrite_or()

```
FormulaT smtrat::JunctorMerger::rewrite_or (
    const FormulaT & formula,
    FormulaSetT && subformulas ) [inline]
```

15.263.1.3 rewrite_xor()

```
FormulaT smtrat::JunctorMerger::rewrite_xor (
    const FormulaT & formula,
    FormulasMultiT && subformulas ) [inline]
```

15.264 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

15.264.1 Detailed Description

Parses keywords: `:simple_symbol`

15.264.2 Constructor & Destructor Documentation

15.264.2.1 KeywordParser()

`smtrat::parser::KeywordParser::KeywordParser ( ) [inline]`

15.264.3 Field Documentation

15.264.3.1 main

`qi::rule<Iterator, std::string(), Skipper> smtrat::parser::KeywordParser↩ ::main`

15.264.3.2 simple

`SimpleSymbolParser smtrat::parser::KeywordParser::simple`

15.265 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)

15.265.1 Constructor & Destructor Documentation

15.265.1.1 lbool() [1/3]

```
Minisat::lbool::lbool (
    uint8_t v ) [inline], [explicit]
```

15.265.1.2 lbool() [2/3]

```
Minisat::lbool::lbool ( ) [inline]
```

15.265.1.3 lbool() [3/3]

```
Minisat::lbool::lbool (
    bool x ) [inline], [explicit]
```

15.265.2 Member Function Documentation

15.265.2.1 operator"!=()

```
bool Minisat::lbool::operator!= (
    lbool b ) const [inline]
```

15.265.2.2 operator&&()

```
lbool Minisat::lbool::operator&& (
    lbool b ) const [inline]
```

15.265.2.3 operator==()

```
bool Minisat::lbool::operator== (
    lbool b ) const [inline]
```

15.265.2.4 operator^()

```
lbool Minisat::lbool::operator^ (
    bool b ) const [inline]
```

15.265.2.5 operator"||()

```
lbool Minisat::lbool::operator|| (
    lbool b ) const [inline]
```

15.265.3 Friends And Related Function Documentation

15.265.3.1 toInt

```
int toInt (
    lbool l ) [friend]
```

15.265.3.2 toLbool

```
lbool toLbool (
    int v ) [friend]
```

15.266 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

15.266.1 Constructor & Destructor Documentation

15.266.1.1 LearnedBound() [1/4]

```
template<class Settings, typename T1, typename T2>
smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound::LearnedBound ( ) [delete]
```

15.266.1.2 LearnedBound() [2/4]

```
template<class Settings, typename T1, typename T2>
smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound::LearnedBound (
            const LearnedBound & ) [delete]
```

15.266.1.3 LearnedBound() [3/4]

```
template<class Settings, typename T1, typename T2>
smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound::LearnedBound (
            LearnedBound && _toMove ) [inline]
```

15.266.1.4 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]
```

15.266.1.5 ~LearnedBound()

```
template<class Settings, typename T1, typename T2>
smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound::~LearnedBound ( ) [inline]
```

15.266.2 Field Documentation

15.266.2.1 newBound

```
template<class Settings, typename T1, typename T2>
const Bound<T1, T2>* smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound::newBound
```

15.266.2.2 newLimit

```
template<class Settings, typename T1, typename T2>
Value<T1> smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound::newLimit
```

15.266.2.3 nextWeakerBound

`template<class Settings, typename T1, typename T2>`
`Bound<T1, T2>::BoundSet::const_iterator smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound::nextWeakerBound`

15.266.2.4 premise

`template<class Settings, typename T1, typename T2>`
`std::vector< const Bound<T1, T2>*> smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound::premise`

15.267 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.

15.267.1 Constructor & Destructor Documentation

15.267.1.1 Lemma()

```
smtrat::Module::Lemma::Lemma (
    const FormulaT & _lemma,
    const LemmaType & _lemmaType,
    const FormulaT & _preferredFormula ) [inline]
```

Constructor.

Parameters

_lemma	The lemma to learn.
_lemmaType	The type of the lemma.
_preferredFormula	The formula within the lemma, which should be assigned to true in the next decision.

15.267.2 Field Documentation

15.267.2.1 mLemma

`FormulaT smtrat::Module::Lemma::mLemma`

The lemma to learn.

15.267.2.2 mLemmaType

`LemmaType smtrat::Module::Lemma::mLemmaType`

The type of the lemma.

15.267.2.3 mPreferredFormula `FormulaT smtrat::Module::Lemma::mPreferredFormula`

The formula within the lemma, which should be assigned to true in the next decision.

15.268 Minisat::LessThan_default< T > Struct Template Reference

`#include <Sort.h>`

Public Member Functions

- bool operator() (T x, T y)

15.268.1 Member Function Documentation

15.268.1.1 operator()() `template<class T >`

```
bool Minisat::LessThan_default< T >::operator() (
            T x,
            T y ) [inline]
```

15.269 smtrat::cad::sample_compare::level Struct Reference

`#include <SampleComparator.h>`

15.270 smtrat::cad::projection_compare::level Struct Reference

`#include <ProjectionComparator.h>`

15.271 smtrat::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.

15.271.1 Member Function Documentation

15.271.1.1 isBound() `bool smtrat::cad::ProjectionLevelInformation::LevelInfo::isBound (`

```
std::size_t pid ) const [inline]
```

15.271.1.2 isEvaluated()

```
bool smtrat::cad::ProjectionLevelInformation::LevelInfo::isEvaluated (
    std::size_t pid ) const [inline]
```

15.271.1.3 isPurged()

```
bool smtrat::cad::ProjectionLevelInformation::LevelInfo::isPurged (
    std::size_t pid ) const [inline]
```

15.271.1.4 removePurgedFromEvaluated()

```
void smtrat::cad::ProjectionLevelInformation::LevelInfo::removePurgedFromEvaluated ( ) [inline]
```

15.271.1.5 restrictEvaluatedToPurged()

```
void smtrat::cad::ProjectionLevelInformation::LevelInfo::restrictEvaluatedToPurged ( ) [inline]
```

15.271.1.6 setBound()

```
void smtrat::cad::ProjectionLevelInformation::LevelInfo::setBound (
    std::size_t pid,
    bool isBound ) [inline]
```

15.271.1.7 setEvaluated()

```
void smtrat::cad::ProjectionLevelInformation::LevelInfo::setEvaluated (
    std::size_t pid,
    bool isEvaluated ) [inline]
```

15.271.1.8 setPurged()

```
void smtrat::cad::ProjectionLevelInformation::LevelInfo::setPurged (
    std::size_t pid,
    bool isPurged ) [inline]
```

15.271.2 Field Documentation

15.271.2.1 bounds

```
carl::Bitset smtrat::cad::ProjectionLevelInformation::LevelInfo::bounds
```

Which polynomials are bounds.

15.271.2.2 ecs

```
EquationalConstraints smtrat::cad::ProjectionLevelInformation::LevelInfo::ecs
```

Equational constraints.

15.271.2.3 evaluated

```
carl::Bitset smtrat::cad::ProjectionLevelInformation::LevelInfo::evaluated
```

Which polynomials have been evaluated w.r.t. purging.

15.271.2.4 purged `carl::Bitset smtrat::cad::ProjectionLevelInformation::LevelInfo::purged`

Which polynomials are purged from the projection. (usually due to bounds)

15.272 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 carl::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 carl::RealAlgebraicPoint< Rational > & point

15.272.1 Member Function Documentation

15.272.1.1 coeffNonNull()

```
std::optional<Poly> smtrat::mcsat::onecellcad::OneCellCAD::coeff↩
NonNull (
            const TagPoly & boundCandidate )  [inline], [inherited]
```

15.272.1.2 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,...,a_i-1,x_i)). Note that this cell is cylindrical only with respect to the given 'variableOrder'.

Parameters

variableOrder	must contain unique variables and at least one, because constant polynomials (without a variable) are prohibited.
point	point.size() >= variables.size().
polys	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

15.272.1.3 isMainPointInsideCell()

```
bool smtrat::mcsat::onecellcad::OneCellCAD::isMainPointInsideCell (
            const CADCell & cell ) [inline], [inherited]
```

15.272.1.4 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(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!).

15.272.1.5 isPointRootOfPoly() [1/2]

```
bool smtrat::mcsat::onecellcad::OneCellCAD::isPointRootOfPoly (
            const std::size_t polyLevel,
            const Poly & poly ) [inline], [inherited]
```

15.272.1.6 isPointRootOfPoly() [2/2]

```
bool smtrat::mcsat::onecellcad::OneCellCAD::isPointRootOfPoly (
            const TagPoly & poly ) [inline], [inherited]
```

15.272.1.7 OneCellCAD()

```
smtrat::mcsat::onecellcad::OneCellCAD::OneCellCAD [inline]
```

15.272.1.8 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'.

15.272.1.9 vanishesEarly()

```
bool smtrat::mcsat::onecellcad::OneCellCAD::vanishesEarly (
            const std::size_t polyLevel,
            const Poly & poly ) [inline], [inherited]
```

Check if an n-variate 'poly' p(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'.

15.272.2 Field Documentation

15.272.2.1 point

```
const carl::RealAlgebraicPoint<Rational>& smtrat::mcsat::onecellcad::One↩
CellCAD::point [inherited]
```

15.272.2.2 variableOrder

```
const std::vector<carl::Variable>& smtrat::mcsat::onecellcad::One↩
CellCAD::variableOrder [inherited]
```

Variables can also be indexed by level.

Polys with mathematical level 1 contain the variable in variableOrder[0]

15.273 smtrat::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 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 ConstraintT &constraint, const std::map< datastructures::SampledDerivation↩Ref< PropSet >, std::vector< ConstraintT >> &derivationConstraints)
- std::vector< ConstraintT > getConstraintsOfCovering (std::map< datastructures::SampledDerivationRef< PropSet >, std::vector< ConstraintT >> &mDerivationToConstraint)
- std::optional< datastructures::SampledDerivationRef< PropSet > > constructDerivation (std::map< datastructures::SampledDerivationRef< PropSet >, std::vector< ConstraintT >> &mDerivationTo↩Constraint)

15.273.1 Constructor & Destructor Documentation

15.273.1.1 LevelWiseInformation() [1/2]

```
template<class Settings>
smtrat::LevelWiseInformation< Settings >::LevelWiseInformation ( ) [inline]
```

15.273.1.2 LevelWiseInformation() [2/2]

```
template<class Settings>
smtrat::LevelWiseInformation< Settings >::LevelWiseInformation (
            LevelWiseInformation< Settings > && other ) [inline]
```

15.273.2 Member Function Documentation

15.273.2.1 addDerivation() [1/2]

```
template<class Settings>
void smtrat::LevelWiseInformation< Settings >::addDerivation (
            const datastructures::SampledDerivationRef< PropSet > & derivation ) [inline]
```

15.273.2.2 addDerivation() [2/2]

```
template<class Settings>
void smtrat::LevelWiseInformation< Settings >::addDerivation (
            datastructures::SampledDerivationRef< PropSet > && derivation ) [inline]
```

15.273.2.3 clear()

```
template<class Settings>
void smtrat::LevelWiseInformation< Settings >::clear ( ) [inline]
```

15.273.2.4 computeCovering()

```
template<class Settings>
bool smtrat::LevelWiseInformation< Settings >::computeCovering ( ) [inline]
```

15.273.2.5 constructDerivation()

```
template<class Settings>
std::optional<datastructures::SampledDerivationRef<PropSet> > smtrat::LevelWiseInformation<
Settings >::constructDerivation (
            std::map< datastructures::SampledDerivationRef< PropSet >, std::vector< ConstraintT
>> & mDerivationToConstraint ) [inline]
```

15.273.2.6 getConstraintsOfCovering()

```
template<class Settings>
std::vector<ConstraintT> smtrat::LevelWiseInformation< Settings >::getConstraintsOfCovering (
            std::map< datastructures::SampledDerivationRef< PropSet >, std::vector< ConstraintT
>> & mDerivationToConstraint ) [inline]
```

15.273.2.7 getCoveringStatus()

```
template<class Settings>
CoveringStatus smtrat::LevelWiseInformation< Settings >::getCoveringStatus ( ) const [inline]
```

15.273.2.8 getDerivations()

```
template<class Settings>
const std::set<datastructures::SampledDerivationRef<PropSet>, SampledDerivationRefCompare>&
smtrat::LevelWiseInformation< Settings >::getDerivations ( ) const [inline]
```

15.273.2.9 getSampleOutside() template<class Settings>

```
const RAN& smtrat::LevelWiseInformation< Settings >::getSampleOutside ( ) const  [inline]
```

15.273.2.10 isFailedCovering() template<class Settings>

```
bool smtrat::LevelWiseInformation< Settings >::isFailedCovering ( ) const  [inline]
```

15.273.2.11 isFullCovering() template<class Settings>

```
bool smtrat::LevelWiseInformation< Settings >::isFullCovering ( ) const  [inline]
```

15.273.2.12 isPartialCovering() template<class Settings>

```
bool smtrat::LevelWiseInformation< Settings >::isPartialCovering ( ) const  [inline]
```

15.273.2.13 isUnknownCovering() template<class Settings>

```
bool smtrat::LevelWiseInformation< Settings >::isUnknownCovering ( ) const  [inline]
```

15.273.2.14 operator=() template<class Settings>

```
LevelWiseInformation& smtrat::LevelWiseInformation< Settings >::operator= (
            LevelWiseInformation< Settings > && other )  [inline]
```

15.273.2.15 removeConstraint() template<class Settings>

```
void smtrat::LevelWiseInformation< Settings >::removeConstraint (
            const ConstraintT & constraint,
            const std::map< datastructures::SampledDerivationRef< PropSet >, std::vector<
ConstraintT >> & derivationConstraints )  [inline]
```

15.273.2.16 removeDerivation() [1/2] template<class Settings>

```
void smtrat::LevelWiseInformation< Settings >::removeDerivation (
            const datastructures::SampledDerivationRef< PropSet > & derivation )  [inline]
```

15.273.2.17 removeDerivation() [2/2] template<class Settings>

```
void smtrat::LevelWiseInformation< Settings >::removeDerivation (
            const std::vector< datastructures::SampledDerivationRef< PropSet >> & derivations
)  [inline]
```

15.273.2.18 setDerivations() template<class Settings>

```
void smtrat::LevelWiseInformation< Settings >::setDerivations (
            std::vector< datastructures::SampledDerivationRef< PropSet >> && derivations )
[inline]
```

15.274 smtrat::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 isConsistent () const
- std::string printSample (Iterator sample) const
- std::string printFullSamples () const

15.274.1 Member Typedef Documentation

15.274.1.1 Constraints `template<typename Settings>`

```
using smtrat::cad::LiftingTree< Settings >::Constraints = CADConstraints<Settings::backtracking>
```

15.274.1.2 FSC `template<typename Settings>`

```
using smtrat::cad::LiftingTree< Settings >::FSC = FullSampleComparator<Iterator,Settings↩
::fullSampleComparator>
```

15.274.1.3 Iterator `template<typename Settings>`

```
using smtrat::cad::LiftingTree< Settings >::Iterator = Tree::iterator
```

15.274.1.4 SC `template<typename Settings>`

```
using smtrat::cad::LiftingTree< Settings >::SC = SampleComparator<Iterator,Settings::sample↩
Comparator>
```

15.274.1.5 Tree template<typename Settings>

```
using smtrat::cad::LiftingTree< Settings >::Tree = carl::tree<Sample>
```

15.274.2 Constructor & Destructor Documentation

15.274.2.1 LiftingTree() [1/3] template<typename Settings>

```
smtrat::cad::LiftingTree< Settings >::LiftingTree (
            const Constraints & c )  [inline]
```

15.274.2.2 LiftingTree() [2/3] template<typename Settings>

```
smtrat::cad::LiftingTree< Settings >::LiftingTree (
            const LiftingTree< Settings > &  )  [delete]
```

15.274.2.3 LiftingTree() [3/3] template<typename Settings>

```
smtrat::cad::LiftingTree< Settings >::LiftingTree (
            LiftingTree< Settings > &&  )  [delete]
```

15.274.3 Member Function Documentation

15.274.3.1 addTrivialSample() template<typename Settings>

```
bool smtrat::cad::LiftingTree< Settings >::addTrivialSample (
            Iterator sample )  [inline]
```

15.274.3.2 extractSampleMap() template<typename Settings>

```
Assignment smtrat::cad::LiftingTree< Settings >::extractSampleMap (
            Iterator it ) const  [inline]
```

15.274.3.3 getLiftingQueue() template<typename Settings>

```
const auto& smtrat::cad::LiftingTree< Settings >::getLiftingQueue ( ) const  [inline]
```

15.274.3.4 getNextFullSample() template<typename Settings>

```
Iterator smtrat::cad::LiftingTree< Settings >::getNextFullSample ( )  [inline]
```

15.274.3.5 getNextSample() template<typename Settings>

```
Iterator smtrat::cad::LiftingTree< Settings >::getNextSample ( )  [inline]
```

15.274.3.6 getTree() template<typename Settings>

```
const auto& smtrat::cad::LiftingTree< Settings >::getTree ( ) const  [inline]
```

15.274.3.7 hasFullSamples() template<typename Settings>

```
bool smtrat::cad::LiftingTree< Settings >::hasFullSamples ( ) const  [inline]
```

15.274.3.8 hasNextSample()

```
template<typename Settings>
bool smtrat::cad::LiftingTree< Settings >::hasNextSample ( ) const [inline]
```

15.274.3.9 isConsistent()

```
template<typename Settings>
bool smtrat::cad::LiftingTree< Settings >::isConsistent ( ) const [inline]
```

15.274.3.10 liftSample()

```
template<typename Settings>
bool smtrat::cad::LiftingTree< Settings >::liftSample (
            Iterator sample,
            const UPoly & p,
            std::size_t pid ) [inline]
```

15.274.3.11 operator=() [1/2]

```
template<typename Settings>
LiftingTree& smtrat::cad::LiftingTree< Settings >::operator= (
            const LiftingTree< Settings > & ) [delete]
```

15.274.3.12 operator=() [2/2]

```
template<typename Settings>
LiftingTree& smtrat::cad::LiftingTree< Settings >::operator= (
            LiftingTree< Settings > && ) [delete]
```

15.274.3.13 printFullSamples()

```
template<typename Settings>
std::string smtrat::cad::LiftingTree< Settings >::printFullSamples ( ) const [inline]
```

15.274.3.14 printSample()

```
template<typename Settings>
std::string smtrat::cad::LiftingTree< Settings >::printSample (
            Iterator sample ) const [inline]
```

15.274.3.15 removedConstraint()

```
template<typename Settings>
void smtrat::cad::LiftingTree< Settings >::removedConstraint (
            const carl::Bitset & mask ) [inline]
```

15.274.3.16 removedPolynomialsFromLevel()

```
template<typename Settings>
void smtrat::cad::LiftingTree< Settings >::removedPolynomialsFromLevel (
            std::size_t level,
            const carl::Bitset & mask ) [inline]
```

15.274.3.17 removeNextSample()

```
template<typename Settings>
void smtrat::cad::LiftingTree< Settings >::removeNextSample ( ) [inline]
```

15.274.3.18 reset() `template<typename Settings>`

```
void smtrat::cad::LiftingTree< Settings >::reset (
            std::vector< carl::Variable > && vars ) [inline]
```

15.274.3.19 resetFullSamples() `template<typename Settings>`

```
void smtrat::cad::LiftingTree< Settings >::resetFullSamples ( ) [inline]
```

15.274.3.20 restoreRemovedSamples() `template<typename Settings>`

```
void smtrat::cad::LiftingTree< Settings >::restoreRemovedSamples ( ) [inline]
```

15.275 smtrat::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

15.275.1 Member Function Documentation

15.275.1.1 initialize() `void smtrat::resource::Limiter::initialize ( ) [inline]`

15.275.1.2 reset() `void smtrat::resource::Limiter::reset ( ) [inline]`

15.275.1.3 resetTimeout() `void smtrat::resource::Limiter::resetTimeout ( ) const [inline]`

15.275.1.4 setMemout()

```
void smtrat::resource::Limiter::setMemout (
            std::size_t megabytes ) [inline]
```

15.275.1.5 setTimeout()

```
void smtrat::resource::Limiter::setTimeout (
            std::size_t seconds ) [inline]
```

15.275.1.6 setTimeoutHandler()

```
void smtrat::resource::Limiter::setTimeoutHandler (
            std::function< void()> f ) [inline]
```

15.275.1.7 timeoutHandler()

```
std::function<void()> smtrat::resource::Limiter::timeoutHandler (
) const [inline]
```

15.276 smtrat::ICPModule< Settings >::linearVariable Struct Reference

`#include <ICPModule.h>`

Data Fields

- FormulaT constraint
- ConstraintT origin

15.276.1 Field Documentation

15.276.1.1 constraint `template<class Settings>`

`FormulaT smtrat::ICPModule< Settings >::linearVariable::constraint`

15.276.1.2 origin `template<class Settings>`

`ConstraintT smtrat::ICPModule< Settings >::linearVariable::origin`

15.277 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)

15.277.1 Constructor & Destructor Documentation

15.277.1.1 Link() [1/2]

```
smtrat::mcsat::ClauseChain::Link::Link (
    const FormulaT && clause,
    const FormulaT && impliedTseitinLiteral ) [inline]
```

15.277.1.2 Link() [2/2]

```
smtrat::mcsat::ClauseChain::Link::Link (
    const FormulaT && clause ) [inline]
```

15.277.2 Member Function Documentation

15.277.2.1 clause()

`const FormulaT& smtrat::mcsat::ClauseChain::Link::clause ( ) const [inline]`

15.277.2.2 impliedTseitinLiteral()

`const FormulaT& smtrat::mcsat::ClauseChain::Link::implied↩TseitinLiteral ( ) const [inline]`

15.277.2.3 isConflicting()

`bool smtrat::mcsat::ClauseChain::Link::isConflicting ( ) const [inline]`

15.277.2.4 isOptional() `bool smtrat::mcsat::ClauseChain::Link::isOptional ( ) const [inline]`

15.277.2.5 isPropagating() `bool smtrat::mcsat::ClauseChain::Link::isPropagating ( ) const [inline]`

15.277.3 Friends And Related Function Documentation

15.277.3.1 operator<< `std::ostream& operator<< ( std::ostream & stream, const Link & link ) [friend]`

15.278 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)

15.278.1 Member Function Documentation

15.278.1.1 operator"!=() `bool Minisat::Lit::operator!= ( Lit p ) const [inline]`

15.278.1.2 operator<() `bool Minisat::Lit::operator< ( Lit p ) const [inline]`

15.278.1.3 operator==() `bool Minisat::Lit::operator== ( Lit p ) const [inline]`

15.278.2 Friends And Related Function Documentation

15.278.2.1 mkLit `Lit mkLit ( Var var, bool sign = false ) [friend]`

15.278.3 Field Documentation

15.278.3.1 x `int Minisat::Lit::x`

15.279 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.

15.279.1 Detailed Description

This backend simply runs files sequentially on the local machine.

15.279.2 Member Function Documentation

15.279.2.1 addResult()

```
void benchmax::Backend::addResult (
    const Tool * tool,
    const fs::path & file,
    BenchmarkResult && result ) [inline], [inherited]
```

Add a result.

15.279.2.2 collect_results()

```
virtual bool benchmax::Backend::collect_results (
    const Jobs & ,
    bool ) [inline], [protected], [virtual], [inherited]
```

15.279.2.3 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.

15.279.2.4 finalize() `virtual void benchmax::Backend::finalize ( ) [inline], [protected], [virtual], [inherited]`

Hook to allow for asynchronous backends to wait for jobs to terminate.

15.279.2.5 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.

15.279.2.6 process_results() `void benchmax::Backend::process_results ( const Jobs & jobs, bool check_finished ) [inline], [inherited]`

15.279.2.7 run() `void benchmax::Backend::run ( const Jobs & jobs, bool wait_for_termination ) [inline], [inherited]`

Run the list of tools against the list of benchmarks.

15.279.2.8 sanitize_results() `void benchmax::Backend::sanitize_results ( const Jobs & jobs ) const [inline], [protected], [inherited]`

15.279.2.9 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.

15.279.2.10 suspendable() `bool benchmax::Backend::suspendable ( ) const [inline], [inherited]`

15.279.2.11 write_results() `void benchmax::Backend::write_results ( const Jobs & jobs ) const [inline], [protected], [inherited]`

15.279.3 Field Documentation

15.279.3.1 mExpectedJobs `std::size_t benchmax::Backend::mExpectedJobs [protected], [inherited]`

Number of jobs that should be run.

15.279.3.2 mFinishedJobs `std::atomic<std::size_t> benchmax::Backend::mFinishedJobs [protected], [inherited]`

Number of jobs that are finished.

15.279.3.3 mLastPercent `std::atomic<std::size_t> benchmax::Backend::mLastPercent [protected], [inherited]`

Percentage of finished jobs when madeProgress() was last called.

15.280 smtrat::LOG Class Reference

`#include <LOG.h>`

Public Member Functions

- bool isDebugEnabled ()

Data Fields

- bool debugEnabled = false

Friends

- class carl::Singleton< LOG >

15.280.1 Member Function Documentation

15.280.1.1 isDebugEnabled() `bool smtrat::LOG::isDebugEnabled ( ) [inline]`

15.280.2 Friends And Related Function Documentation

15.280.2.1 carl::Singleton< LOG > `friend class carl::Singleton< LOG > [friend]`

15.280.3 Field Documentation

15.280.3.1 debugEnabled `bool smtrat::LOG::debugEnabled = false`

15.281 smtrat::parser::LogicParser Struct Reference

`#include <Script.h>`

Public Member Functions

- LogicParser ()

15.281.1 Constructor & Destructor Documentation

15.281.1.1 LogicParser() `smtrat::parser::LogicParser::LogicParser ( ) [inline]`

15.282 smtrat::LongFormulaEncoder Class Reference

`#include <LongFormulaEncoder.h>`

Public Member Functions

- LongFormulaEncoder ()
- bool canEncode (const ConstraintT &constraint)
- Rational encodingSize (const ConstraintT &constraint)
- std::string name ()
- boost::optional< FormulaT > encode (const ConstraintT &constraint)

  Encodes an arbitrary constraint.

Data Fields

- std::size_t problem_size

Protected Member Functions

- boost::optional< FormulaT > doEncode (const ConstraintT &constraint)
- FormulaT generateVarChain (const std::set< carl::Variable > &vars, carl::FormulaType type)

15.282.1 Constructor & Destructor Documentation

15.282.1.1 LongFormulaEncoder() `smtrat::LongFormulaEncoder::LongFormulaEncoder ( ) [inline]`

15.282.2 Member Function Documentation

15.282.2.1 canEncode() `bool smtrat::LongFormulaEncoder::canEncode ( const ConstraintT & constraint ) [virtual]`

Implements smtrat::PseudoBoolEncoder.

15.282.2.2 doEncode() `boost::optional< FormulaT > smtrat::LongFormulaEncoder::doEncode ( const ConstraintT & constraint ) [protected], [virtual]`

Implements smtrat::PseudoBoolEncoder.

15.282.2.3 encode() `boost::optional< FormulaT > smtrat::PseudoBoolEncoder::encode ( const ConstraintT & constraint ) [inherited]`

Encodes an arbitrary constraint.

Returns encoded formula

15.282.2.4 encodingSize() `Rational smtrat::LongFormulaEncoder::encodingSize ( const ConstraintT & constraint ) [virtual]`

Reimplemented from smtrat::PseudoBoolEncoder.

15.282.2.5 generateVarChain() `FormulaT smtrat::PseudoBoolEncoder::generateVarChain ( const std::set< carl::Variable > & vars, carl::FormulaType type ) [protected], [inherited]`

15.282.2.6 name() `std::string smtrat::LongFormulaEncoder::name ( ) [inline], [virtual]`

Reimplemented from smtrat::PseudoBoolEncoder.

15.282.3 Field Documentation

15.282.3.1 problem_size `std::size_t smtrat::PseudoBoolEncoder::problem_size [inherited]`

15.283 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 lra::Bound< LRABoundType, LRAEntryType > LRABound

  Type of a bound of a variable within the LRAModule.
- typedef lra::Variable< LRABoundType, LRAEntryType > LRAVariable

  A variable of the LRAModule, being either a original variable or a slack variable representing a linear polynomial.
- typedef lra::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 lra::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 LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::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 EvalRationalMap & getRationalModel () const

  Gives a rational model if the received formula is satisfiable.

- Answer optimize (Answer _result)
- Answer checkNotEqualConstraints (Answer _result)
- void processLearnedBounds ()
- void createInfeasibleSubsets (lra::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 lra 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 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 _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

- 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 &_branching↩ Value) 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 _preferLeft↩ Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩ ::vector< FormulaT >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩ AsPremise=false)

- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩CaseWeak=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.

15.283.1 Detailed Description

template<class Settings>
class smtrat::LRAModule< Settings >

A module which performs the Simplex method on the linear part of it's received formula.

15.283.2 Member Typedef Documentation

15.283.2.1 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)

15.283.2.2 ConstraintContextMap

```
template<class Settings>
typedef carl::FastMap<FormulaT, Context> smtrat::LRAModule< Settings >::ConstraintContextMap
```

Stores the position of a received formula in the passed formula.

15.283.2.3 ExVariableMap

```
template<class Settings>
typedef carl::FastPointerMap<typename Poly::PolyType, LRAVariable*> smtrat::LRAModule< Settings >::ExVariableMap
```

Maps a linear polynomial to it's corresponding LRAModule variable.

15.283.2.4 LRABound

```
template<class Settings>
typedef lra::Bound<LRABoundType, LRAEntryType> smtrat::LRAModule< Settings >::LRABound
```

Type of a bound of a variable within the LRAModule.

15.283.2.5 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.

15.283.2.6 LRAEntryType template<class Settings>

```
typedef Settings::EntryType smtrat::LRAModule< Settings >::LRAEntryType
```

Type of an entry within the tableau.

15.283.2.7 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.

15.283.2.8 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 value of the bound type.

15.283.2.9 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.

15.283.2.10 SettingsType template<class Settings>

```
typedef Settings smtrat::LRAModule< Settings >::SettingsType
```

15.283.2.11 VarVariableMap template<class Settings>

```
typedef carl::FastMap<carl::Variable, LRAVariable*> smtrat::LRAModule< Settings >::VarVariableMap
```

Maps an original variable to it's corresponding LRAModule variable.

15.283.3 Member Enumeration Documentation

15.283.3.1 LemmaType enum smtrat::Module::LemmaType : unsigned [strong], [inherited]

Enumerator

NORMAL	
PERMANENT	

15.283.4 Constructor & Destructor Documentation

15.283.4.1 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 LRAModule.
_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.

15.283.4.2 ~LRAModule() template<class Settings >

```
smtrat::LRAModule< Settings >::~LRAModule ( ) [virtual]
```

Destructs this LRAModule.

15.283.5 Member Function Documentation

15.283.5.1 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.

15.283.5.2 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

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

15.283.5.3 addCore()

```
template<class Settings >
bool smtrat::LRAModule< 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.

15.283.5.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
            const FormulaT & formula )  [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.283.5.5 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.

15.283.5.6 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.

15.283.5.7 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.

15.283.5.8 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.

15.283.5.9 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

_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.

15.283.5.10 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

_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.

15.283.5.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns All satisfying assignments, if existent.

15.283.5.12 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.

15.283.5.13 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.

15.283.5.14 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.

15.283.5.15 branchAt() [1/6]

```
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]
```

15.283.5.16 branchAt() [2/6]

```
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]
```

15.283.5.17 branchAt() [3/6]

```
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]
```

15.283.5.18 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.

Parameters

_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.283.5.19 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            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]
```

15.283.5.20 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

15.283.5.21 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in smtrat::PModule.

15.283.5.22 checkCore()

```
template<class Settings >
Answer smtrat::LRAModule< 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.

15.283.5.23 checkCore_old()

```
template<class Settings>
Answer smtrat::LRAModule< Settings >::checkCore_old ( )
```

15.283.5.24 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
    std::vector< FormulaSetT >::const_iterator _infsubset,
    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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.283.5.25 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.

15.283.5.26 checkNotEqualConstraints()

```
template<class Settings >
Answer smtrat::LRAModule< Settings >::checkNotEqualConstraints (
    Answer _result )
```

15.283.5.27 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.283.5.28 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.283.5.29 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.283.5.30 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.283.5.31 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.283.5.32 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

15.283.5.33 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.

15.283.5.34 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.283.5.35 constraintsToInform()

```
const carl::FastSet<FormulaT>& smtrat::Module::constraints↩
ToInform ( ) const  [inline], [inherited]
```

Returns The constraints which the backends must be informed about.

15.283.5.36 createInfeasibleSubsets()

```
template<class Settings >
void smtrat::LRAModule< Settings >::createInfeasibleSubsets (
            lra::EntryID _tableauEntry )
```

15.283.5.37 currentlySatisfied()

```
template<class Settings >
unsigned smtrat::LRAModule< Settings >::currentlySatisfied (
            const FormulaT & _formula ) 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.

15.283.5.38 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)

15.283.5.39 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

_constraint	The constraint to remove from internal data structures.

15.283.5.40 deinformCore()

```
template<class Settings >
void smtrat::LRAModule< 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

_constraint	The constraint to remove from internal data structures.

Reimplemented from smtrat::Module.

15.283.5.41 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.283.5.42 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.283.5.43 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.283.5.44 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.283.5.45 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.

15.283.5.46 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.

15.283.5.47 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

15.283.5.48 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

15.283.5.49 getBackendsAllModels()

`void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

15.283.5.50 getBackendsModel()

`void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

15.283.5.51 getInfeasibleSubsets() [1/2]

`void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

15.283.5.52 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.283.5.53 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.283.5.54 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.

15.283.5.55 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.283.5.56 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.283.5.57 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.283.5.58 getRationalModel()

```
template<class Settings >
const EvalRationalMap & smtrat::LRAModule< 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.

15.283.5.59 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 reveived formula, if the Boolean is true.

Reimplemented in smtrat::PModule.

15.283.5.60 getSlackVariable()

```
template<class Settings>
const LRAVariable* smtrat::LRAModule< Settings >::getSlackVariable (
    const FormulaT & _constraint ) const [inline]
```

Parameters

_constraint	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.

15.283.5.61 getVariableBounds() `template<class Settings >`
`EvalRationalIntervalMap smtrat::LRAModule< Settings >::getVariableBounds ( ) const`

Returns the bounds of the variables as intervals.

Returns The bounds of the variables as intervals.

15.283.5.62 hasLemmas() `bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

15.283.5.63 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).

15.283.5.64 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns A unique ID to identify this module instance.

15.283.5.65 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.

15.283.5.66 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.283.5.67 informBackends()

```
void smtrat::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.

15.283.5.68 informCore()

```
template<class Settings >
bool smtrat::LRAModule< 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

_constraint	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.

15.283.5.69 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.283.5.70 init()

```
template<class Settings >
void smtrat::LRAModule< Settings >::init ( ) [virtual]
```

Initializes the tableau according to all linear constraints, of which this module has been informed.

Reimplemented from smtrat::Module.

15.283.5.71 is_minimizing()

```
bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]
```

15.283.5.72 isLemmaLevel()

```
bool smtrat::Module::isLemmaLevel (
    LemmaLevel level ) [protected], [inherited]
```

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.283.5.73 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().

15.283.5.74 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.

15.283.5.75 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.283.5.76 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.283.5.77 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.

15.283.5.78 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.

15.283.5.79 objective()

```
carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]
```

15.283.5.80 optimize()

```
template<class Settings >
Answer smtrat::LRAModule< Settings >::optimize (
            Answer _result )
```

15.283.5.81 originalVariables()

```
template<class Settings>
const VarVariableMap& smtrat::LRAModule< Settings >::originalVariables ( ) const [inline]
```

Returns A constant reference to the original variables.

15.283.5.82 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
            const FormulaT & _origin ) const [protected], [inherited]
```

15.283.5.83 passedFormulaBegin()

```
ModuleInput::iterator smtrat::Module::passedFormulaBegin ( )
[inline], [protected], [inherited]
```

Returns An iterator to the end of the passed formula. TODO: disable this method

15.283.5.84 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula.

15.283.5.85 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to the backends of this module.

15.283.5.86 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to this module.

15.283.5.87 print()

```
void smtrat::Module::print (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.283.5.88 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

_out	The stream to print the assignment on.

15.283.5.89 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

_out	The output stream to print on.
_init	The beginning of each line to print.

15.283.5.90 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

_out	The output stream to print on.
_init	The beginning of each line to print.

15.283.5.91 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
            const std::string & _initiation = "***" ) const  [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.283.5.92 printLinearConstraints()

```
template<class Settings>
void smtrat::LRAModule< Settings >::printLinearConstraints (
            std::ostream & _out = std::cout,
            const std::string _init = "" ) const
```

Prints all linear constraints.

Parameters

_out	The output stream to print on.
_init	The beginning of each line to print.

15.283.5.93 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

_out	The stream to print the assignment on.

15.283.5.94 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

_out	The output stream to print on.
_init	The beginning of each line to print.

15.283.5.95 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
            const std::string & _initiation = "***" ) const  [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.283.5.96 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

_out	The output stream to print on.
_init	The beginning of each line to print.

15.283.5.97 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
            const std::string & _initiation = "***" ) const  [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.283.5.98 printTableau()

```
template<class Settings>
void smtrat::LRAModule< Settings >::printTableau (
            std::ostream & _out = std::cout,
            const std::string _init = "" ) const
```

Prints the current tableau.

Parameters

_out	The output stream to print on.
_init	The beginning of each line to print.

15.283.5.99 printVariables()

```
template<class Settings>
void smtrat::LRAModule< Settings >::printVariables (
            std::ostream & _out = std::cout,
            const std::string _init = "" ) const
```

Prints all lra variables and their assignments.

Parameters

_out	The output stream to print on.
_init	The beginning of each line to print.

15.283.5.100 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.

15.283.5.101 processLearnedBounds() template<class Settings >

void smtrat::LRAModule< Settings >::processLearnedBounds ( )

15.283.5.102 processResult() template<class Settings >

Answer smtrat::LRAModule< Settings >::processResult (
Answer _result)

15.283.5.103 receivedFormulaChecked() void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]

Notifies that the received formulas has been checked.

15.283.5.104 receivedFormulasAsInfeasibleSubset() void smtrat::Module::receivedFormulasAs↩InfeasibleSubset (

ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]

Stores an infeasible subset consisting only of the given received formula.

15.283.5.105 receivedVariable() bool smtrat::Module::receivedVariable (

carl::Variable::Arg _var) const [inline], [inherited]

15.283.5.106 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.

15.283.5.107 removeCore() template<class Settings >

void smtrat::LRAModule< Settings >::removeCore (
ModuleInput::const_iterator _subformula) [virtual]

Removes everything related to the given sub-formula of the received formula.

Parameters

_subformula	The sub formula of the received formula to remove.

Reimplemented from smtrat::Module.

15.283.5.108 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.283.5.109 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]
```

15.283.5.110 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.283.5.111 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.283.5.112 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]
```

Reimplemented in smtrat::PModule.

15.283.5.113 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

_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 in smtrat::PModule.

15.283.5.114 setId()

`void smtrat::Module::setId ( std::size_t _id ) [inline], [inherited]`

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.283.5.115 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.

15.283.5.116 slackVariables()

`template<class Settings> const ExVariableMap& smtrat::LRAModule< Settings >::slackVariables ( ) const [inline]`

Returns A constant reference to the slack variables.

15.283.5.117 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.

15.283.5.118 splitUnequalConstraint()

`void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.283.5.119 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.283.5.120 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 >.

15.283.5.121 updateLemmas()

`void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.283.5.122 updateModel()

`template<class Settings > void smtrat::LRAModule< Settings >::updateModel ( ) const [virtual]`

Updates the model, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented from smtrat::Module.

15.283.5.123 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).

15.283.6 Field Documentation

15.283.6.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.283.6.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.283.6.3 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.

15.283.6.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.283.6.5 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

15.283.6.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.283.6.7 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.

15.283.6.8 mFirstUncheckedReceivedSubformula `ModuleInput::const_iterator smtrat::Module::m↩FirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

15.283.6.9 mFoundAnswer `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.283.6.10 mFullCheck `bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.283.6.11 mInfeasibleSubsets `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.283.6.12 mInformedConstraints `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.283.6.13 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.

15.283.6.14 mLemmas `std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.283.6.15 mModel `Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.283.6.16 mModelComputed `bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.283.6.17 mNumOfBranchVarsToStore `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.283.6.18 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.

15.283.6.19 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

15.283.6.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.283.6.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.283.6.22 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.

15.283.6.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

15.283.6.24 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).

15.283.6.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.284 smtrat::LVEModule< Settings > Class Template Reference

`#include <LVEModule.h>`

Public Types

- typedef Settings SettingsType
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::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 smtrat::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 _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.

- 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 &_branching↩ Value) 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 _preferLeft↩ Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩ ::vector< FormulaT >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩ AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩ CaseWeak=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.

15.284.1 Member Typedef Documentation

15.284.1.1 SettingsType `template<typename Settings >`

```
typedef Settings smtrat::LVEModule< Settings >::SettingsType
```

15.284.2 Member Enumeration Documentation

15.284.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

15.284.3 Constructor & Destructor Documentation

15.284.3.1 LVEModule() `template<class Settings >`

```
smtrat::LVEModule< Settings >::LVEModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

15.284.3.2 ~LVEModule() `template<class Settings >`

```
smtrat::LVEModule< Settings >::~LVEModule ( )
```

15.284.4 Member Function Documentation

15.284.4.1 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

_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.

15.284.4.2 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 >.

15.284.4.3 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

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 smtrat::SATModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::VSModule< Settings >, smtrat::PFEModule< Settings >, smtrat::EQModule< Setting smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Sett smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, and smtrat::GBModule< Settings >.

15.284.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.284.4.5 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.

15.284.4.6 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.

15.284.4.7 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.

15.284.4.8 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.

15.284.4.9 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.

15.284.4.10 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

_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.

15.284.4.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns All satisfying assignments, if existent.

15.284.4.12 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.

15.284.4.13 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.

15.284.4.14 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.

15.284.4.15 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.

15.284.4.16 branchAt() [1/6]

```
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]
```

15.284.4.17 branchAt() [2/6]

```
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]
```

15.284.4.18 branchAt() [3/6]

```
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]
```

15.284.4.19 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.284.4.20 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            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]
```

15.284.4.21 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

15.284.4.22 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

15.284.4.23 checkCore()

```
template<class Settings >
Answer smtrat::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 smtrat::Module.

15.284.4.24 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
    std::vector< FormulaSetT >::const_iterator _infsubset,
    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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.284.4.25 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.

15.284.4.26 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.284.4.27 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.284.4.28 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.284.4.29 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.284.4.30 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.284.4.31 collectOrigins() [1/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

15.284.4.32 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.

15.284.4.33 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.284.4.34 constraintsToInform() `const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.284.4.35 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

15.284.4.36 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)

15.284.4.37 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

_constraint	The constraint to remove from internal data structures.

15.284.4.38 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.284.4.39 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.284.4.40 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.284.4.41 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceived↩
VariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.284.4.42 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBest↩
Origin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.284.4.43 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.

15.284.4.44 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.

15.284.4.45 freeSplittingVariable() `static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

15.284.4.46 generateTrivialInfeasibleSubset() `void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

15.284.4.47 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

15.284.4.48 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

15.284.4.49 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

15.284.4.50 getInfeasibleSubsets() [2/2] `std::vector< FormulaSetT > smtrat::Module::getInfeasible↩ Subsets ( 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.284.4.51 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.284.4.52 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.

15.284.4.53 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.284.4.54 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.284.4.55 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.284.4.56 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 reveived formula, if the Boolean is true.

Reimplemented from smtrat::Module.

15.284.4.57 hasLemmas() bool smtrat::Module::hasLemmas ( ) [inline], [inherited]

Checks whether this module or any of its backends provides any lemmas.

15.284.4.58 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).

15.284.4.59 id() std::size_t smtrat::Module::id ( ) const [inline], [inherited]

Returns A unique ID to identify this module instance.

15.284.4.60 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.

15.284.4.61 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.284.4.62 informBackends()

```
void smtrat::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.

15.284.4.63 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, and smtrat::ICPModule< ICPSettings1 >.

15.284.4.64 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.284.4.65 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::FPPModule< Settings >.

15.284.4.66 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.284.4.67 isLemmaLevel() `bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.284.4.68 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().

15.284.4.69 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.

15.284.4.70 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.284.4.71 model()

`const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.284.4.72 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.

15.284.4.73 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.

15.284.4.74 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

15.284.4.75 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
    const FormulaT & _origin ) const [protected], [inherited]
```

15.284.4.76 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula. TODO: disable this method

15.284.4.77 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula.

15.284.4.78 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to the backends of this module.

15.284.4.79 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to this module.

15.284.4.80 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.284.4.81 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

_out	The stream to print the assignment on.

15.284.4.82 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.284.4.83 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

_out	The stream to print the assignment on.

15.284.4.84 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.284.4.85 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.284.4.86 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.

15.284.4.87 receivedFormulaChecked() `void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

15.284.4.88 receivedFormulasAsInfeasibleSubset() `void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

15.284.4.89 receivedVariable() `bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

15.284.4.90 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

_subformula	The sub formula of the received formula to remove.

Reimplemented from smtrat::Module.

15.284.4.91 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

formula	The sub formula of the received formula to remove.

Reimplemented in smtrat::SATModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::VSModule< Settings >, smtrat::PFEModule< Settings >, smtrat::EQModule< Setting smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Sett smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, and smtrat::GBModule< Settings >.

15.284.4.92 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.284.4.93 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

15.284.4.94 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline],
[inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.284.4.95 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.284.4.96 runBackends() [1/2]

```
virtual Answer smtrat::PModule::runBackends ( ) [inline],
[virtual], [inherited]
```

Reimplemented from smtrat::Module.

15.284.4.97 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.

15.284.4.98 setId()

```
void smtrat::Module::setId (
    std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ ↩ *id*	The id to set this module's id to.

15.284.4.99 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.

15.284.4.100 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.

15.284.4.101 splitUnequalConstraint()

```
void smtrat::Module::splitUnequalConstraint (
    const FormulaT & _unequalConstraint ) [protected], [inherited]
```

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.284.4.102 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.284.4.103 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 >.

15.284.4.104 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.284.4.105 updateModel() `template<class Settings > void smtrat::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 smtrat::Module.

15.284.4.106 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).

15.284.5 Field Documentation

15.284.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.284.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.284.5.3 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.

15.284.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.284.5.5 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

15.284.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.284.5.7 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.

15.284.5.8 mFirstUncheckedReceivedSubformula `ModuleInput::const_iterator smtrat::Module::m↩ FirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

15.284.5.9 mFoundAnswer `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.284.5.10 mFullCheck `bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.284.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.284.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.284.5.13 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.

15.284.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.284.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.284.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.284.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.284.5.18 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.

15.284.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplitting↩Variables [static], [inherited]`

Reusable splitting variables.

15.284.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.284.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.284.5.22 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.

15.284.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

15.284.5.24 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).

15.284.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.285 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

  Determines variables assigned by the currently found satisfying assignment to an equal value in their domain.

- const Model & model () const
- 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

15.285.1 Detailed Description

Base class for solvers.

This is the interface to the user.

15.285.2 Constructor & Destructor Documentation

15.285.2.1 Manager()

`smtrat::Manager::Manager ( )`

Constructs a manager.

15.285.2.2 ~Manager()

`smtrat::Manager::~Manager ( )`

Destructs a manager.

15.285.3 Member Function Documentation

15.285.3.1 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

_subformula	The formula to add.
_containsUnknownConstraints	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.

15.285.3.2 addBackend() [1/2]

```
template<typename Module >
BackendLink smtrat::Manager::addBackend (
            const BackendLink & backend )  [inline], [protected]
```

15.285.3.3 addBackend() [2/2]

```
template<typename Module >
BackendLink smtrat::Manager::addBackend (
            const std::initializer_list< BackendLink > & backends = {} )  [inline], [protected]
```

15.285.3.4 addEdge()

```
BackendLink& smtrat::Manager::addEdge (
            std::size_t from,
            std::size_t to )  [inline], [protected]
```

15.285.3.5 addInformationRelevantFormula()

```
void smtrat::Manager::addInformationRelevantFormula
(
            const FormulaT & formula )  [inline]
```

Adds formula to InformationRelevantFormula.

Parameters

formula	Formula to add

15.285.3.6 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.

15.285.3.7 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.

15.285.3.8 clear() `void smtrat::Manager::clear ( ) [inline]`

15.285.3.9 clearInformationRelevantFormula() `void smtrat::Manager::clearInformationRelevant↩ Formula ( ) [inline]`

Deletes all InformationRelevantFormula.

15.285.3.10 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

_constraint	The constraint to remove from internal data structures.

15.285.3.11 formula() `const ModuleInput& smtrat::Manager::formula ( ) const [inline]`

Returns The conjunction of so far added formulas.

15.285.3.12 formulaBegin() `ModuleInput::iterator smtrat::Manager::formulaBegin ( ) [inline]`

15.285.3.13 formulaEnd() `ModuleInput::iterator smtrat::Manager::formulaEnd ( ) [inline]`

15.285.3.14 getAllBackends() `std::vector<Module*> smtrat::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

_module	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.

15.285.3.15 getAllGeneratedModules()

```
const std::vector<Module*>& smtrat::Manager::getAll↩
GeneratedModules ( ) const [inline]
```

Returns All instantiated modules of the solver belonging to this manager.

15.285.3.16 getBackends()

```
std::vector< Module * > smtrat::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

_requiredBy	The module asking for a backend.
_foundAnswer	A conditional

Returns A vector of modules, which the module defined by _requiredBy calls in parallel to achieve an answer to the given instance.

15.285.3.17 getInformationRelevantFormulas()

```
const std::set<FormulaT>& smtrat::Manager::get↩
InformationRelevantFormulas ( ) [inline], [protected]
```

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.285.3.18 getInputSimplified()

```
std::pair< bool, FormulaT > smtrat::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.

15.285.3.19 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.

15.285.3.20 infeasibleSubsets()

```
const std::vector< FormulaSetT > & smtrat::Manager::infeasible↩
Subsets ( ) 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.

15.285.3.21 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

_constraint	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.

15.285.3.22 isLemmaLevel()

`bool smtrat::Manager::isLemmaLevel ( LemmaLevel level ) [inline]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.285.3.23 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.

15.285.3.24 logic()

`auto smtrat::Manager::logic ( ) const [inline]`

Returns A constant reference to the logic this solver considers.

15.285.3.25 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.

15.285.3.26 objectiveVariable()

`const carl::Variable& smtrat::Manager::objectiveVariable ( ) const [inline]`

15.285.3.27 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.

15.285.3.28 pop() [2/2] void smtrat::Manager::pop (size_t _levels)

15.285.3.29 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

_out	The stream to print on.

15.285.3.30 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.

15.285.3.31 printBackTrackStack() void smtrat::Manager::printBackTrackStack (std::ostream & _out = std::cout) const

Prints the stack of backtrack points.

Parameters

_out	The stream to print on.

15.285.3.32 printInfeasibleSubset() void smtrat::Manager::printInfeasibleSubset (std::ostream & _out = std::cout) const

Prints the first found infeasible subset of the set of received formulas.

Parameters

_out	The stream to print on.

15.285.3.33 printStrategyGraph()

```
void smtrat::Manager::printStrategyGraph (
    std::ostream & _out = std::cout ) const [inline]
```

Prints the strategy of the solver maintained by this manager.

Parameters

_out	The stream to print on.

15.285.3.34 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.

15.285.3.35 rDebugOutputChannel()

```
std::ostream& smtrat::Manager::rDebugOutputChannel ( ) [inline]
```

Returns The stream to print the debug information on.

15.285.3.36 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

_subformula	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.

15.285.3.37 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

_subformula	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.

15.285.3.38 reset() `void smtrat::Manager::reset ( )`

15.285.3.39 rLogic() `auto& smtrat::Manager::rLogic ( ) [inline]`

Returns A reference to the logic this solver considers.

15.285.3.40 setLemmaLevel() `void smtrat::Manager::setLemmaLevel ( LemmaLevel level ) [inline]`

Sets the current level for lemma generation.

Parameters

level	Level

15.285.3.41 setObjectiveVariable() `void smtrat::Manager::setObjectiveVariable ( carl::Variable var ) [inline]`

15.285.3.42 setStrategy() [1/2] `void smtrat::Manager::setStrategy ( const BackendLink & backend ) [inline], [protected]`

15.285.3.43 setStrategy() [2/2] `void smtrat::Manager::setStrategy ( const std::initializer_list< BackendLink > & backends ) [inline], [protected]`

15.285.4 Friends And Related Function Documentation

15.285.4.1 Module `friend class Module [friend]`

15.286 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

15.286.1 Constructor & Destructor Documentation

15.286.1.1 Map() [1/2]

```
template<class K, class D, class H = Hash<K>, class E = Equal<K>>
Minisat::Map< K, D, H, E >::Map ( ) [inline]
```

15.286.1.2 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]
```

15.286.1.3 ~Map()

```
template<class K, class D, class H = Hash<K>, class E = Equal<K>>
Minisat::Map< K, D, H, E >::~Map ( ) [inline]
```

15.286.2 Member Function Documentation

15.286.2.1 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]
```

15.286.2.2 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]
```

15.286.2.3 clear()

```
template<class K, class D, class H = Hash<K>, class E = Equal<K>>
void Minisat::Map< K, D, H, E >::clear ( ) [inline]
```

15.286.2.4 elems()

```
template<class K, class D, class H = Hash<K>, class E = Equal<K>>
int Minisat::Map< K, D, H, E >::elems ( ) const [inline]
```

15.286.2.5 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]
```

15.286.2.6 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]
```

15.286.2.7 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]
```

15.286.2.8 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]
```

15.286.2.9 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]
```

15.286.2.10 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]
```

15.286.2.11 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]
```

15.287 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.

15.287.1 Detailed Description

Tool wrapper for MathSAT for SMT-LIB.

15.287.2 Constructor & Destructor Documentation

15.287.2.1 MathSAT()

```
benchmax::MathSAT::MathSAT (
    const fs::path & binary,
    const std::string & arguments ) [inline]
```

New MathSAT tool.

15.287.3 Member Function Documentation

15.287.3.1 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.

15.287.3.2 attributeHash() `std::size_t benchmax::Tool::attributeHash ( ) const [inline], [inherited]`

Hash of the attributes.

15.287.3.3 attributes() `const std::map<std::string,std::string>& benchmax::Tool::attributes ( ) const [inline], [inherited]`

A set of attributes, for example compilation options.

15.287.3.4 binary() `fs::path benchmax::Tool::binary ( ) const [inline], [inherited]`

Full path to the binary.

15.287.3.5 canHandle() `virtual bool benchmax::MathSAT::canHandle ( const fs::path & path ) const [inline], [override], [virtual]`

Only handle .smt2 files.

Reimplemented from benchmax::Tool.

15.287.3.6 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.

15.287.3.7 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.

15.287.3.8 name() `std::string benchmax::Tool::name ( ) const [inline], [inherited]`

Common name of this tool.

15.287.3.9 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.

15.287.3.10 resolveDependencies() `std::vector<std::string> benchmax::Tool::resolveDependencies ( ) const [inline], [inherited]`

Get dependencies of binary required to run it (via ldd)

15.287.4 Field Documentation

15.287.4.1 mArguments `std::string benchmax::Tool::mArguments [protected], [inherited]`

Command line arguments that should be passed to the binary.

15.287.4.2 mAttributes `std::map<std::string,std::string> benchmax::Tool::mAttributes [protected], [inherited]`

Attributes of the tool obtained by introspection of the binary.

15.287.4.3 mBinary `fs::path benchmax::Tool::mBinary [protected], [inherited]`

Path to the binary.

15.287.4.4 mName `std::string benchmax::Tool::mName [protected], [inherited]`

(Non-unique) identifier for the tool.

15.288 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

15.288.1 Detailed Description

This heuristic chooses the explanation excluding the largest interval.

15.288.2 Member Function Documentation

15.288.2.1 operator()() `bool smtrat::mcsat::fm::MaxSizeComparator::operator() ( const Bound & b1, const Bound & b2 ) const [inline]`

15.288.3 Field Documentation

15.288.3.1 symmetric `bool smtrat::mcsat::fm::MaxSizeComparator::symmetric = true`

15.289 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 ()

15.289.1 Constructor & Destructor Documentation

15.289.1.1 MaxSMT()

```
template<typename Solver, MaxSMTStrategy Strategy>
smtrat::MaxSMT< Solver, Strategy >::MaxSMT (
            Solver & s )  [inline]
```

15.289.2 Member Function Documentation

15.289.2.1 active()

```
template<typename Solver, MaxSMTStrategy Strategy>
bool smtrat::MaxSMT< Solver, Strategy >::active ( ) const  [inline]
```

15.289.2.2 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]
```

15.289.2.3 compute()

```
template<typename Solver, MaxSMTStrategy Strategy>
std::tuple<Answer,Model,ObjectiveValues> smtrat::MaxSMT< Solver, Strategy >::compute ( )
[inline]
```

15.289.2.4 remove_soft_formula()

```
template<typename Solver, MaxSMTStrategy Strategy>
void smtrat::MaxSMT< Solver, Strategy >::remove_soft_formula (
            const FormulaT & formula )  [inline]
```

15.289.2.5 reset()

```
template<typename Solver, MaxSMTStrategy Strategy>
void smtrat::MaxSMT< Solver, Strategy >::reset ( )  [inline]
```

15.290 smtrat::maxsmt::MaxSMTBackend< Solver, Strategy > Class Template Reference

```
#include <MaxSMT.h>
```

15.291 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 ()

15.291.1 Constructor & Destructor Documentation

15.291.1.1 MaxSMTBackend() template<typename Solver >

```
smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::FU_MALIK_INCREMENTAL >::MaxSMTBackend (
            Solver & solver,
            const std::vector< FormulaT > & softClauses )  [inline]
```

15.291.2 Member Function Documentation

15.291.2.1 run() template<typename Solver >

```
Answer smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::FU_MALIK_INCREMENTAL >::run ( )
[inline]
```

15.292 smtrat::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 ()

15.292.1 Constructor & Destructor Documentation

15.292.1.1 MaxSMTBackend() template<typename Solver >

```
smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::LINEAR_SEARCH >::MaxSMTBackend (
            Solver & solver,
            const std::vector< FormulaT > & softClauses )  [inline]
```

15.292.2 Member Function Documentation

15.292.2.1 run() template<typename Solver >

```
Answer smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::LINEAR_SEARCH >::run ( )  [inline]
```

15.293 smtrat::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 ()

15.293.1 Constructor & Destructor Documentation

15.293.1.1 MaxSMTBackend() template<typename Solver >

```
smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::MSU3 >::MaxSMTBackend (
            Solver & solver,
            const std::vector< FormulaT > & softClauses )  [inline]
```

15.293.2 Member Function Documentation

15.293.2.1 run() template<typename Solver >

```
Answer smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::MSU3 >::run ( )  [inline]
```

15.294 smtrat::MCBModule< Settings > Class Template Reference

```
#include <MCBModule.h>
```

Public Types

- typedef Settings SettingsType
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::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 smtrat::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 _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.

- 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 &_branching↩Value) 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 _preferLeft↩Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩::vector< FormulaT >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩CaseWeak=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.

15.294.1 Member Typedef Documentation

15.294.1.1 SettingsType

```
template<typename Settings >
typedef Settings smtrat::MCBModule< Settings >::SettingsType
```

15.294.2 Member Enumeration Documentation

15.294.2.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

15.294.3 Constructor & Destructor Documentation

15.294.3.1 MCBModule()

```
template<class Settings >
smtrat::MCBModule< Settings >::MCBModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

15.294.3.2 ~MCBModule()

```
template<class Settings >
smtrat::MCBModule< Settings >::~MCBModule ( )
```

15.294.4 Member Function Documentation

15.294.4.1 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

_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.

15.294.4.2 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 >.

15.294.4.3 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

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 smtrat::SATModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::VSModule< Settings >, smtrat::PFEModule< Settings >, smtrat::EQModule< Setting smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Sett smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, and smtrat::GBModule< Settings >.

15.294.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.294.4.5 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.

15.294.4.6 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.

15.294.4.7 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.

15.294.4.8 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.

15.294.4.9 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.

15.294.4.10 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

_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.

15.294.4.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const  [inline],
[inherited]
```

Returns All satisfying assignments, if existent.

15.294.4.12 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.

15.294.4.13 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.

15.294.4.14 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.

15.294.4.15 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.

15.294.4.16 branchAt() [1/6]

```
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]
```

15.294.4.17 branchAt() [2/6]

```
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]
```

15.294.4.18 branchAt() [3/6]

```
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]
```

15.294.4.19 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.294.4.20 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
    const typename P::PolyType & _poly,
    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]
```

15.294.4.21 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false )  [inline], [protected], [inherited]
```

15.294.4.22 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

15.294.4.23 checkCore()

```
template<class 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.

15.294.4.24 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
            std::vector< FormulaSetT >::const_iterator _infsubset,
            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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.294.4.25 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.

15.294.4.26 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.294.4.27 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.294.4.28 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.294.4.29 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.294.4.30 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.294.4.31 collectOrigins() [1/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

15.294.4.32 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.

15.294.4.33 collectTheoryPropagations()

`void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.294.4.34 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ToInform ( ) const [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.294.4.35 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

15.294.4.36 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)

15.294.4.37 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

_constraint	The constraint to remove from internal data structures.

15.294.4.38 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.294.4.39 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.294.4.40 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.294.4.41 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceived↩
VariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.294.4.42 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.294.4.43 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.

15.294.4.44 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.

15.294.4.45 freeSplittingVariable()

```
static void smtrat::Module::freeSplittingVariable (
    const FormulaT & _splittingVariable ) [inline], [static], [inherited]
```

15.294.4.46 generateTrivialInfeasibleSubset()

```
void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]
```

Stores the trivial infeasible subset being the set of received formulas.

15.294.4.47 getBackendsAllModels()

```
void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]
```

Stores all models of a backend in the list of all models of this module.

15.294.4.48 getBackendsModel()

```
void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]
```

15.294.4.49 getInfeasibleSubsets() [1/2]

```
void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]
```

Copies the infeasible subsets of the passed formula.

15.294.4.50 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.294.4.51 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.294.4.52 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.

15.294.4.53 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.294.4.54 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.294.4.55 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.294.4.56 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 reveived formula, if the Boolean is true.

Reimplemented from smtrat::Module.

15.294.4.57 hasLemmas() bool smtrat::Module::hasLemmas ( ) [inline], [inherited]

Checks whether this module or any of its backends provides any lemmas.

15.294.4.58 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).

15.294.4.59 id() std::size_t smtrat::Module::id ( ) const [inline], [inherited]

Returns A unique ID to identify this module instance.

15.294.4.60 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.

15.294.4.61 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.294.4.62 informBackends()

```
void smtrat::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.

15.294.4.63 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, and smtrat::ICPModule< ICPSettings1 >.

15.294.4.64 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.294.4.65 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::FPPModule< Settings >.

15.294.4.66 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.294.4.67 isLemmaLevel() `bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.294.4.68 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().

15.294.4.69 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.

15.294.4.70 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.294.4.71 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.294.4.72 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.

15.294.4.73 moduleName() `virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]`

Returns The name of the given module type as name.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::EQPreprocessingModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::VSModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::GBModule< Settings >, smtrat::IntEqModule< Setti smtrat::CubeLIAModule< Settings >, smtrat::CurryModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IncWidthModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::BEModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::SplitSOSModule< Settings >.

15.294.4.74 objective() `carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

15.294.4.75 originInReceivedFormula() `bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

15.294.4.76 passedFormulaBegin() `ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula. TODO: disable this method

15.294.4.77 passedFormulaEnd() `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula.

15.294.4.78 pPassedFormula() `const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to the backends of this module.

15.294.4.79 pReceivedFormula() `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to this module.

15.294.4.80 print() `void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.294.4.81 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

_out	The stream to print the assignment on.

15.294.4.82 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.294.4.83 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

_out	The stream to print the assignment on.

15.294.4.84 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.294.4.85 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.294.4.86 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.

15.294.4.87 receivedFormulaChecked()

```
void smtrat::Module::receivedFormulaChecked ( )  [inline], [inherited]
```

Notifies that the received formulas has been checked.

15.294.4.88 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩InfeasibleSubset (
    ModuleInput::const_iterator _subformula )  [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

15.294.4.89 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const  [inline], [inherited]
```

15.294.4.90 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

_subformula	The sub formula of the received formula to remove.

Reimplemented from smtrat::Module.

15.294.4.91 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

formula	The sub formula of the received formula to remove.

Reimplemented in smtrat::SATModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::VSModule< Settings >, smtrat::PFEModule< Settings >, smtrat::EQModule< Setting smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Sett smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, and smtrat::GBModule< Settings >.

15.294.4.92 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin
(
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.294.4.93 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

15.294.4.94 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline],
[inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.294.4.95 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.294.4.96 runBackends() [1/2]

`virtual Answer smtrat::PModule::runBackends ( ) [inline], [virtual], [inherited]`

Reimplemented from smtrat::Module.

15.294.4.97 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.

15.294.4.98 setId()

```
void smtrat::Module::setId (
    std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.294.4.99 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.

15.294.4.100 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.

15.294.4.101 splitUnequalConstraint()

`void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.294.4.102 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.294.4.103 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 >.

15.294.4.104 updateLemmas()

`void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.294.4.105 updateModel()

`template<typename Settings >`
`void smtrat::MCBModule< Settings >::updateModel ( ) const [virtual]`

Updates the model, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented from smtrat::Module.

15.294.4.106 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).

15.294.5 Field Documentation

15.294.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.294.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.294.5.3 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.

15.294.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.294.5.5 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

15.294.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.294.5.7 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.

15.294.5.8 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.

15.294.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.294.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.294.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.294.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.294.5.13 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.

15.294.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.294.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.294.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.294.5.17 mNumOfBranchVarsToStore `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.294.5.18 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.

15.294.5.19 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

15.294.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.294.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.294.5.22 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.

15.294.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

15.294.5.24 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).

15.294.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.295 smtrat::mcsat::MCSATBackend< 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
- template<typename Constraints >
  void initVariables (const Constraints &c)
- 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

15.295.1 Member Function Documentation

15.295.1.1 assignedVariables()

```
template<typename Settings>
const auto& smtrat::mcsat::MCSATBackend< Settings >::assignedVariables ( ) const  [inline]
```

15.295.1.2 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]
```

15.295.1.3 explain() [2/2]

```
template<typename Settings>
Explanation smtrat::mcsat::MCSATBackend< Settings >::explain (
            carl::Variable var,
            const FormulaT & f,
            const FormulasT & reason )  [inline]
```

15.295.1.4 findAssignment()

```
template<typename Settings>
AssignmentOrConflict smtrat::mcsat::MCSATBackend< Settings >::findAssignment (
            carl::Variable var ) const  [inline]
```

15.295.1.5 getModel()

```
template<typename Settings>
const auto& smtrat::mcsat::MCSATBackend< Settings >::getModel ( ) const  [inline]
```

15.295.1.6 getTrail()

```
template<typename Settings>
const auto& smtrat::mcsat::MCSATBackend< Settings >::getTrail ( ) const  [inline]
```

15.295.1.7 initVariables() template<typename Settings>

```
template<typename Constraints >
void smtrat::mcsat::MCSATBackend< Settings >::initVariables (
            const Constraints & c )  [inline]
```

15.295.1.8 isActive() template<typename Settings>

```
bool smtrat::mcsat::MCSATBackend< Settings >::isActive (
            const FormulaT & f ) const  [inline]
```

15.295.1.9 isInfeasible() template<typename Settings>

```
AssignmentOrConflict smtrat::mcsat::MCSATBackend< Settings >::isInfeasible (
            carl::Variable var,
            const FormulaT & f )  [inline]
```

15.295.1.10 popAssignment() template<typename Settings>

```
void smtrat::mcsat::MCSATBackend< Settings >::popAssignment (
            carl::Variable v )  [inline]
```

15.295.1.11 popConstraint() template<typename Settings>

```
void smtrat::mcsat::MCSATBackend< Settings >::popConstraint (
            const FormulaT & f )  [inline]
```

15.295.1.12 pushAssignment() template<typename Settings>

```
void smtrat::mcsat::MCSATBackend< Settings >::pushAssignment (
            carl::Variable v,
            const ModelValue & mv,
            const FormulaT & f )  [inline]
```

15.295.1.13 pushConstraint() template<typename Settings>

```
void smtrat::mcsat::MCSATBackend< Settings >::pushConstraint (
            const FormulaT & f )  [inline]
```

15.295.1.14 variables() template<typename Settings>

```
const auto& smtrat::mcsat::MCSATBackend< Settings >::variables ( ) const  [inline]
```

15.296 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::lbool 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 isConsistent ()

  Checks if any inconsistency was detected.
- bool isConsistent (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
- 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 assignedAtTrailIndex (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)

15.296.1 Constructor & Destructor Documentation

15.296.1.1 MCSATMixin() `template<typename Settings>`

```
template<typename BaseModule >
smtrat::mcsat::MCSATMixin< Settings >::MCSATMixin (
            BaseModule & baseModule )  [inline], [explicit]
```

15.296.2 Member Function Documentation

15.296.2.1 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.

15.296.2.2 assignedAtTrailIndex() `template<typename Settings>`

```
int smtrat::mcsat::MCSATMixin< Settings >::assignedAtTrailIndex (
            Minisat::Var variable ) const  [inline]
```

15.296.2.3 backtrackTo() `template<typename Settings>`

```
bool smtrat::mcsat::MCSATMixin< Settings >::backtrackTo (
            Minisat::Lit literal )
```

Backtracks to the theory decision represented by the given literal.

15.296.2.4 carlVar() `template<typename Settings>`

```
const carl::Variable& smtrat::mcsat::MCSATMixin< Settings >::carlVar (
            Minisat::Var v ) const  [inline]
```

15.296.2.5 computeTheoryLevel() [1/2] `template<typename Settings>`

```
std::size_t smtrat::mcsat::MCSATMixin< Settings >::computeTheoryLevel (
            const FormulaT & f ) const  [inline]
```

15.296.2.6 computeTheoryLevel() [2/2] `template<typename Settings>`

```
std::size_t smtrat::mcsat::MCSATMixin< Settings >::computeTheoryLevel (
            Minisat::Var var ) const  [inline]
```

15.296.2.7 current() [1/2]

```
template<typename Settings>
TheoryLevel& smtrat::mcsat::MCSATMixin< Settings >::current ( ) [inline]
```

15.296.2.8 current() [2/2]

```
template<typename Settings>
const TheoryLevel& smtrat::mcsat::MCSATMixin< Settings >::current ( ) const [inline]
```

Returns the current theory level.

15.296.2.9 decisionLevel()

```
template<typename Settings>
int smtrat::mcsat::MCSATMixin< Settings >::decisionLevel (
            Minisat::Var v ) const [inline]
```

15.296.2.10 doBooleanAssignment()

```
template<typename Settings>
void smtrat::mcsat::MCSATMixin< Settings >::doBooleanAssignment (
            Minisat::Lit lit ) [inline]
```

Add a new constraint.

15.296.2.11 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.

15.296.2.12 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.

15.296.2.13 explainTheoryPropagation()

```
template<typename Settings>
Explanation smtrat::mcsat::MCSATMixin< Settings >::explainTheoryPropagation (
            Minisat::Lit literal ) [inline]
```

15.296.2.14 fullConsistencyCheck()

```
template<typename Settings>
bool smtrat::mcsat::MCSATMixin< Settings >::fullConsistencyCheck ( ) const [inline]
```

15.296.2.15 get() template<typename Settings>

```
const TheoryLevel& smtrat::mcsat::MCSATMixin< Settings >::get (
            std::size_t level ) const  [inline]
```

Returns the respective theory level.

15.296.2.16 getDecisionLiteral() template<typename Settings>

```
Minisat::Lit smtrat::mcsat::MCSATMixin< Settings >::getDecisionLiteral (
            Minisat::Var var ) const  [inline]
```

15.296.2.17 initVariables() template<typename Settings>

```
template<typename Constraints >
void smtrat::mcsat::MCSATMixin< Settings >::initVariables (
            const Constraints & c )  [inline]
```

15.296.2.18 isAssignedTheoryVariable() template<typename Settings>

```
bool smtrat::mcsat::MCSATMixin< Settings >::isAssignedTheoryVariable (
            const carl::Variable & var ) const  [inline]
```

15.296.2.19 isBooleanDecisionFeasible() template<typename Settings>

```
std::pair<bool, std::optional<Explanation> > smtrat::mcsat::MCSATMixin< Settings >::is↩
BooleanDecisionFeasible (
            Minisat::Lit lit,
            bool always_explain = false )
```

15.296.2.20 isConsistent() [1/2] template<typename Settings>

```
bool smtrat::mcsat::MCSATMixin< Settings >::isConsistent ( )  [inline]
```

Checks if any inconsistency was detected.

15.296.2.21 isConsistent() [2/2] template<typename Settings>

```
bool smtrat::mcsat::MCSATMixin< Settings >::isConsistent (
            Minisat::Var v )  [inline]
```

15.296.2.22 isFeasible() template<typename Settings>

```
std::optional<Explanation> smtrat::mcsat::MCSATMixin< Settings >::isFeasible (
            const carl::Variable & var )  [inline]
```

15.296.2.23 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.

15.296.2.24 isTheoryVar()

```
template<typename Settings>
bool smtrat::mcsat::MCSATMixin< Settings >::isTheoryVar (
            Minisat::Var v ) const  [inline]
```

15.296.2.25 level()

```
template<typename Settings>
std::size_t smtrat::mcsat::MCSATMixin< Settings >::level ( ) const  [inline]
```

15.296.2.26 makeTheoryDecision()

```
template<typename Settings>
std::variant<Explanation,FormulasT> smtrat::mcsat::MCSATMixin< Settings >::makeTheoryDecision
(
            const carl::Variable & var )  [inline]
```

15.296.2.27 maxDegree()

```
template<typename Settings>
std::size_t smtrat::mcsat::MCSATMixin< Settings >::maxDegree (
            const Minisat::Var & var )  [inline]
```

15.296.2.28 minisatVar()

```
template<typename Settings>
Minisat::Var smtrat::mcsat::MCSATMixin< Settings >::minisatVar (
            const carl::Variable & v ) const  [inline]
```

15.296.2.29 model()

```
template<typename Settings>
const Model& smtrat::mcsat::MCSATMixin< Settings >::model ( ) const  [inline]
```

15.296.2.30 printClause()

```
template<typename Settings>
void smtrat::mcsat::MCSATMixin< Settings >::printClause (
            std::ostream & os,
            Minisat::CRef clause ) const
```

Prints a single clause.

15.296.2.31 propagateBooleanDomain()

```
template<typename Settings>
std::pair<boost::tribool, std::optional<Explanation> > smtrat::mcsat::MCSATMixin< Settings
>::propagateBooleanDomain (
            Minisat::Lit lit )
```

15.296.2.32 pushTheoryDecision()

```
template<typename Settings>
void smtrat::mcsat::MCSATMixin< Settings >::pushTheoryDecision (
            const FormulaT & assignment,
            Minisat::Lit decisionLiteral )
```

Push a theory decision.

15.296.2.33 theoryAssignmentComplete() template<typename Settings>

```
bool smtrat::mcsat::MCSATMixin< Settings >::theoryAssignmentComplete ( ) const  [inline]
```

15.296.2.34 theoryLevel() template<typename Settings>

```
std::size_t smtrat::mcsat::MCSATMixin< Settings >::theoryLevel (
            Minisat::Var var ) const  [inline]
```

15.296.2.35 theoryVarAbstractions() template<typename Settings>

```
std::vector<Minisat::Var> smtrat::mcsat::MCSATMixin< Settings >::theoryVarAbstractions ( )
const  [inline]
```

15.296.2.36 theoryVars() template<typename Settings>

```
std::vector<Minisat::Var> smtrat::mcsat::MCSATMixin< Settings >::theoryVars (
            const Minisat::Var & var )  [inline]
```

15.296.2.37 topSemanticPropagation() template<typename Settings>

```
Minisat::Var smtrat::mcsat::MCSATMixin< Settings >::topSemanticPropagation ( )  [inline]
```

Getter for semantic propagations.

15.296.2.38 undecidedBooleanVariables() template<typename Settings>

```
const std::vector<Minisat::Var>& smtrat::mcsat::MCSATMixin< Settings >::undecidedBoolean↩
Variables ( ) const  [inline]
```

15.296.2.39 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().

15.296.2.40 variable() template<typename Settings>

```
carl::Variable smtrat::mcsat::MCSATMixin< Settings >::variable (
            std::size_t level ) const  [inline]
```

Retrieve already decided theory variables.

15.296.3 Friends And Related Function Documentation

15.296.3.1 operator<< template<typename Settings>

```
template<typename Sett >
std::ostream& operator<< (
            std::ostream & os,
            const MCSATMixin< Sett > & mcm )  [friend]
```

15.297 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

15.297.1 Member Function Documentation

15.297.1.1 operator()() [1/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const BinaryExpression & expr ) const  [inline], [inherited]
```

15.297.1.2 operator()() [2/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const carl::Variable & expr ) const  [inline], [inherited]
```

15.297.1.3 operator()() [3/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const ExpressionContent * _ec ) const  [inline], [inherited]
```

15.297.1.4 operator()() [4/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const ITEExpression & expr ) const  [inline], [inherited]
```

15.297.1.5 operator()() [5/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const NaryExpression & expr ) const  [inline], [inherited]
```

15.297.1.6 operator()() [6/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const QuantifierExpression & expr ) const  [inline], [inherited]
```

15.297.1.7 operator()() [7/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const UnaryExpression & expr ) const [inline], [inherited]
```

15.297.1.8 simplify() [1/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const BinaryExpression & ) const [inline], [protected], [virtual], [inherited]
```

15.297.1.9 simplify() [2/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const carl::Variable & ) const [inline], [protected], [virtual], [inherited]
```

15.297.1.10 simplify() [3/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const ITEExpression & ) const [inline], [protected], [virtual], [inherited]
```

15.297.1.11 simplify() [4/6]

```
const ExpressionContent* smtrat::expression::simplifier::MergeSimplifier::simplify (
            const NaryExpression & expr ) const [inline], [virtual]
```

Reimplemented from smtrat::expression::simplifier::BaseSimplifier.

15.297.1.12 simplify() [5/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const QuantifierExpression & ) const [inline], [protected], [virtual], [inherited]
```

15.297.1.13 simplify() [6/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const UnaryExpression & ) const [inline], [protected], [virtual], [inherited]
```

Reimplemented in smtrat::expression::simplifier::NegationSimplifier.

15.298 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.

15.298.1 Detailed Description

Tool wrapper for a Minisat solver.

15.298.2 Constructor & Destructor Documentation

15.298.2.1 Minisat()

```
benchmax::Minisat::Minisat (
    const fs::path & binary,
    const std::string & arguments ) [inline]
```

Create tool.

15.298.3 Member Function Documentation

15.298.3.1 additionalResults()

```
virtual void benchmax::Minisat::additionalResults (
    const fs::path & ,
    BenchmarkResult & result ) const [inline], [override], [virtual]
```

Parse results from stdout.

Reimplemented from benchmax::Tool.

15.298.3.2 attributeHash() `std::size_t benchmax::Tool::attributeHash ( ) const [inline], [inherited]`

Hash of the attributes.

15.298.3.3 attributes() `const std::map<std::string,std::string>& benchmax::Tool::attributes ( ) const [inline], [inherited]`

A set of attributes, for example compilation options.

15.298.3.4 binary() `fs::path benchmax::Tool::binary ( ) const [inline], [inherited]`

Full path to the binary.

15.298.3.5 canHandle() `virtual bool benchmax::Minisat::canHandle ( const fs::path & path ) const [inline], [override], [virtual]`

Only handles .opb files.

Reimplemented from benchmax::Tool.

15.298.3.6 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.

15.298.3.7 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.

15.298.3.8 name() `std::string benchmax::Tool::name ( ) const [inline], [inherited]`

Common name of this tool.

15.298.3.9 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.

15.298.3.10 resolveDependencies() `std::vector<std::string> benchmax::Tool::resolveDependencies ( ) const [inline], [inherited]`

Get dependencies of binary required to run it (via ldd)

15.298.4 Field Documentation

15.298.4.1 mArguments `std::string benchmax::Tool::mArguments [protected], [inherited]`

Command line arguments that should be passed to the binary.

15.298.4.2 mAttributes `std::map<std::string,std::string> benchmax::Tool::mAttributes [protected], [inherited]`

Attributes of the tool obtained by introspection of the binary.

15.298.4.3 mBinary `fs::path benchmax::Tool::mBinary [protected], [inherited]`

Path to the binary.

15.298.4.4 mName `std::string benchmax::Tool::mName [protected], [inherited]`

(Non-unique) identifier for the tool.

15.299 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.

15.299.1 Detailed Description

Tool wrapper for the Minisatp solver for pseudo-Boolean problems.

15.299.2 Constructor & Destructor Documentation

15.299.2.1 Minisatp()

```
benchmax::Minisatp::Minisatp (
    const fs::path & binary,
    const std::string & arguments ) [inline]
```

Create tool, add "-v0" to arguments.

15.299.3 Member Function Documentation

15.299.3.1 additionalResults()

```
virtual void benchmax::Minisatp::additionalResults (
    const fs::path & ,
    BenchmarkResult & result ) const [inline], [override], [virtual]
```

Parse results from stdout.

Reimplemented from benchmax::Tool.

15.299.3.2 attributeHash()

```
std::size_t benchmax::Tool::attributeHash ( ) const [inline], [inherited]
```

Hash of the attributes.

15.299.3.3 attributes()

```
const std::map<std::string,std::string>& benchmax::Tool::attributes ( )
const [inline], [inherited]
```

A set of attributes, for example compilation options.

15.299.3.4 binary()

```
fs::path benchmax::Tool::binary ( ) const [inline], [inherited]
```

Full path to the binary.

15.299.3.5 canHandle()

```
virtual bool benchmax::Minisatp::canHandle (
    const fs::path & path ) const [inline], [override], [virtual]
```

Only handles .opb files.

Reimplemented from benchmax::Tool.

15.299.3.6 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.

15.299.3.7 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.

15.299.3.8 name()

```
std::string benchmax::Tool::name ( ) const [inline], [inherited]
```

Common name of this tool.

15.299.3.9 parseCommandline()

```
virtual std::optional<std::string> benchmax::Tool::parse↩
Commandline (
    const std::string & cmdline ) const [inline], [virtual], [inherited]
```

Compose commandline for this tool and the given input file.

15.299.3.10 resolveDependencies()

```
std::vector<std::string> benchmax::Tool::resolveDependencies
( ) const [inline], [inherited]
```

Get dependencies of binary required to run it (via ldd)

15.299.4 Field Documentation

15.299.4.1 mArguments

```
std::string benchmax::Tool::mArguments [protected], [inherited]
```

Command line arguments that should be passed to the binary.

15.299.4.2 mAttributes

```
std::map<std::string,std::string> benchmax::Tool::mAttributes [protected],
[inherited]
```

Attributes of the tool obtained by introspection of the binary.

15.299.4.3 mBinary

```
fs::path benchmax::Tool::mBinary [protected], [inherited]
```

Path to the binary.

15.299.4.4 mName

```
std::string benchmax::Tool::mName [protected], [inherited]
```

(Non-unique) identifier for the tool.

15.300 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

15.300.1 Detailed Description This heuristic chooses the explanation excluding the smallest interval.

15.300.2 Member Function Documentation

15.300.2.1 operator()()

```
bool smtrat::mcsat::fm::MinSizeComparator::operator() (
    const Bound & b1,
    const Bound & b2 ) const  [inline]
```

15.300.3 Field Documentation

15.300.3.1 symmetric

```
bool smtrat::mcsat::fm::MinSizeComparator::symmetric = true
```

15.301 smtrat::mcsat::fm::MinVarCountComparator Struct Reference

This heuristic tries to minimize the number of variables occuring in the explanation.

```
#include <ConflictGenerator.h>
```

Public Member Functions

- bool operator() (const Bound &b1, const Bound &b2) const

Data Fields

- bool symmetric = false

15.301.1 Detailed Description This heuristic tries to minimize the number of variables occuring in the explanation.

It is a 2-approximation to the lowest possible number of variables in an explanation.

15.301.2 Member Function Documentation

15.301.2.1 operator()()

```
bool smtrat::mcsat::fm::MinVarCountComparator::operator() (
    const Bound & b1,
    const Bound & b2 ) const  [inline]
```

15.301.3 Field Documentation

15.301.3.1 symmetric

```
bool smtrat::mcsat::fm::MinVarCountComparator::symmetric = false
```

15.302 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)
- template<>
  void operator() (const CAD &cad, std::vector< FormulaSetT > &mis)
- template<>
  void operator() (const CAD &cad, std::vector< FormulaSetT > &mis)
- template<>
  void operator() (const CAD &cad, std::vector< FormulaSetT > &mis)
- template<>
  void operator() (const CAD &cad, std::vector< FormulaSetT > &mis)
- template<>
  void operator() (const CAD &cad, std::vector< FormulaSetT > &mis)
- template<>
  void operator() (const CAD &cad, std::vector< FormulaSetT > &mis)

15.302.1 Member Function Documentation

15.302.1.1 operator()() [1/7] template<MISHeuristic heuristic>

```
template<typename CAD >
void smtrat::cad::MISGeneration< heuristic >::operator() (
            const CAD & cad,
            std::vector< FormulaSetT > & mis )
```

15.302.1.2 operator()() [2/7] template<>

```
void smtrat::cad::MISGeneration< MISHeuristic::TRIVIAL >::operator() (
            const CAD & cad,
            std::vector< FormulaSetT > & mis )
```

15.302.1.3 operator()() [3/7] template<>

```
void smtrat::cad::MISGeneration< MISHeuristic::GREEDY >::operator() (
            const CAD & cad,
            std::vector< FormulaSetT > & mis )
```

15.302.1.4 operator()() [4/7] template<>

```
void smtrat::cad::MISGeneration< MISHeuristic::GREEDY_PRE >::operator() (
            const CAD & cad,
            std::vector< FormulaSetT > & mis )
```

15.302.1.5 operator()() [5/7] template<>

```
void smtrat::cad::MISGeneration< MISHeuristic::EXACT >::operator() (
            const CAD & cad,
            std::vector< FormulaSetT > & mis )
```

15.302.1.6 operator()() [6/7] template<>

```
void smtrat::cad::MISGeneration< MISHeuristic::HYBRID >::operator() (
            const CAD & cad,
            std::vector< FormulaSetT > & mis )
```

15.302.1.7 operator()() [7/7] template<>

```
void smtrat::cad::MISGeneration< MISHeuristic::GREEDY_WEIGHTED >::operator() (
            const CAD & cad,
            std::vector< FormulaSetT > & mis )
```

15.303 smtrat::MixedSignEncoder Class Reference

```
#include <MixedSignEncoder.h>
```

Public Member Functions

- MixedSignEncoder ()
- bool canEncode (const ConstraintT &constraint)
- Rational encodingSize (const ConstraintT &constraint)
- std::string name ()
- boost::optional< FormulaT > encode (const ConstraintT &constraint)

  Encodes an arbitrary constraint.

Data Fields

- std::size_t problem_size

Protected Member Functions

- boost::optional< FormulaT > doEncode (const ConstraintT &constraint)
- FormulaT generateVarChain (const std::set< carl::Variable > &vars, carl::FormulaType type)

15.303.1 Constructor & Destructor Documentation

15.303.1.1 MixedSignEncoder() smtrat::MixedSignEncoder::MixedSignEncoder ( ) [inline]

15.303.2 Member Function Documentation

15.303.2.1 canEncode() bool smtrat::MixedSignEncoder::canEncode (

```
            const ConstraintT & constraint ) [virtual]
```

Implements smtrat::PseudoBoolEncoder.

15.303.2.2 doEncode() boost::optional< FormulaT > smtrat::MixedSignEncoder::doEncode (

```
            const ConstraintT & constraint ) [protected], [virtual]
```

Implements smtrat::PseudoBoolEncoder.

15.303.2.3 encode() boost::optional< FormulaT > smtrat::PseudoBoolEncoder::encode (

```
            const ConstraintT & constraint ) [inherited]
```

Encodes an arbitrary constraint.

Returns encoded formula

15.303.2.4 encodingSize() Rational smtrat::MixedSignEncoder::encodingSize (

```
            const ConstraintT & constraint ) [virtual]
```

Reimplemented from smtrat::PseudoBoolEncoder.

15.303.2.5 generateVarChain()

```
FormulaT smtrat::PseudoBoolEncoder::generateVarChain (
    const std::set< carl::Variable > & vars,
    carl::FormulaType type ) [protected], [inherited]
```

15.303.2.6 name()

```
std::string smtrat::MixedSignEncoder::name ( ) [inline], [virtual]
```

Reimplemented from smtrat::PseudoBoolEncoder.

15.303.3 Field Documentation

15.303.3.1 problem_size

```
std::size_t smtrat::PseudoBoolEncoder::problem_size [inherited]
```

15.304 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 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

- 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.

15.304.1 Member Typedef Documentation

15.304.1.1 Constraints `template<typename Settings>`

```
using smtrat::cad::BaseProjection< Settings >::Constraints = CADConstraints<Settings::backtracking>
[protected], [inherited]
```

15.304.2 Member Function Documentation

15.304.2.1 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.

15.304.2.2 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 >.

15.304.2.3 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.

15.304.2.4 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::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Incrementality::NONE smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, smtrat::cad::Projection< Incrementality::FULL, BT, S and smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >.

15.304.2.5 callRemoveCallback() template<typename Settings>

```
void smtrat::cad::BaseProjection< Settings >::callRemoveCallback (
            std::size_t level,
            const SampleLiftedWith & slw ) const [inline], [protected], [inherited]
```

15.304.2.6 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.

15.304.2.7 dim() template<typename Settings>

```
std::size_t smtrat::cad::BaseProjection< Settings >::dim ( ) const [inline], [inherited]
```

Returns the dimension of the projection.

15.304.2.8 empty() [1/2] template<typename Settings>

```
virtual bool smtrat::cad::BaseProjection< Settings >::empty ( ) [inline], [virtual], [inherited]
```

15.304.2.9 empty() [2/2] template<typename Settings>

```
virtual bool smtrat::cad::BaseProjection< Settings >::empty (
            std::size_t level ) const [pure virtual], [inherited]
```

Implemented in smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, smtrat::cad::Projection< Incremental smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Increme smtrat::qe::cad::Projection< Settings >, smtrat::qe::cad::Projection< smtrat::qe::cad::CADSettings >, and smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >.

15.304.2.10 exportAsDot() template<typename Settings>

```
virtual void smtrat::cad::BaseProjection< Settings >::exportAsDot (
            std::ostream &  ) const  [inline], [virtual], [inherited]
```

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, and smtrat::cad::Projection< Incre

15.304.2.11 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.

15.304.2.12 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.

15.304.2.13 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 >.

15.304.2.14 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.

15.304.2.15 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::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, smtrat::cad::Projection< Incremental smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Increme smtrat::qe::cad::Projection< Settings >, smtrat::qe::cad::Projection< smtrat::qe::cad::CADSettings >, and smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >.

15.304.2.16 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::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, smtrat::cad::Projection< Incremental smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Increme smtrat::qe::cad::Projection< Settings >, smtrat::qe::cad::Projection< smtrat::qe::cad::CADSettings >, and smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >.

15.304.2.17 isPurged()

```
template<typename Settings>
bool smtrat::cad::BaseProjection< Settings >::isPurged (
            std::size_t level,
            std::size_t id )  [inline], [protected], [inherited]
```

15.304.2.18 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.

15.304.2.19 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::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Incrementality::FULL smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >, and smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >.

15.304.2.20 reset()

```
template<typename Settings>
void smtrat::cad::BaseProjection< Settings >::reset ( )  [inline], [inherited]
```

Resets all datastructures, use the given variables from now on.

15.304.2.21 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.

15.304.2.22 size() [1/2] template<typename Settings>

```
std::size_t smtrat::cad::BaseProjection< Settings >::size ( ) const [inline], [inherited]
```

15.304.2.23 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::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, smtrat::cad::Projection< Incremental smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Increme smtrat::qe::cad::Projection< Settings >, smtrat::qe::cad::Projection< smtrat::qe::cad::CADSettings >, and smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >.

15.304.2.24 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.

15.304.2.25 vars() template<typename Settings>

```
const auto& smtrat::cad::BaseProjection< Settings >::vars ( ) const [inline], [inherited]
```

Returns the variables used for projection.

15.304.3 Field Documentation

15.304.3.1 mConstraints template<typename Settings>

```
const Constraints& smtrat::cad::BaseProjection< Settings >::mConstraints [protected], [inherited]
```

15.304.3.2 mInfo template<typename Settings>

```
ProjectionInformation smtrat::cad::BaseProjection< Settings >::mInfo [protected], [inherited]
```

Additional info on projection, projection levels and projection polynomials.

15.304.3.3 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.

15.304.3.4 mOperator template<typename Settings>

```
ProjectionOperator smtrat::cad::BaseProjection< Settings >::mOperator [protected], [inherited]
```

The projection operator.

15.304.3.5 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.

15.305 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 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 ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, S > &p)

15.305.1 Detailed Description

template<typename Settings>
class smtrat::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.

15.305.2 Constructor & Destructor Documentation

15.305.2.1 ModelBasedProjection()

```
template<typename Settings >
smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >↩
::ModelBasedProjection (
            const Constraints & c,
            const Model & model )  [inline]
```

15.305.3 Member Function Documentation

15.305.3.1 addEqConstraint() [1/2]

```
template<typename Settings>
carl::Bitset smtrat::cad::BaseProjection< Settings >::addEqConstraint (
            const Poly & p,
            std::size_t cid,
            bool isBound )  [inline], [inherited]
```

Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.

15.305.3.2 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], [inherited]
```

Adds the given polynomial of an equational constraint to the projection.

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >.

15.305.3.3 addPolynomial() [1/2]

```
template<typename Settings>
carl::Bitset smtrat::cad::BaseProjection< Settings >::addPolynomial (
            const Poly & p,
            std::size_t cid,
            bool isBound )  [inline], [inherited]
```

Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.

15.305.3.4 addPolynomial() [2/2]

```
template<typename Settings >
carl::Bitset smtrat::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 smtrat::cad::BaseProjection< Settings >.

15.305.3.5 callRemoveCallback() template<typename Settings>

```
void smtrat::cad::BaseProjection< Settings >::callRemoveCallback (
            std::size_t level,
            const SampleLiftedWith & slw ) const [inline], [protected], [inherited]
```

15.305.3.6 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.

15.305.3.7 dim() [1/2] template<typename Settings >

```
std::size_t smtrat::cad::BaseProjection< Settings >::dim [inline]
```

Returns the dimension of the projection.

15.305.3.8 dim() [2/2] template<typename Settings>

```
std::size_t smtrat::cad::BaseProjection< Settings >::dim ( ) const [inline], [inherited]
```

Returns the dimension of the projection.

15.305.3.9 empty() [1/2] template<typename Settings>

```
virtual bool smtrat::cad::BaseProjection< Settings >::empty ( ) [inline], [virtual], [inherited]
```

15.305.3.10 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 >.

15.305.3.11 exportAsDot() template<typename Settings>

```
virtual void smtrat::cad::BaseProjection< Settings >::exportAsDot (
            std::ostream & ) const [inline], [virtual], [inherited]
```

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, and smtrat::cad::Projection< Incre

15.305.3.12 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.

15.305.3.13 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.

15.305.3.14 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 >.

15.305.3.15 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.

15.305.3.16 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 >.

15.305.3.17 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 >.

15.305.3.18 isPurged() template<typename Settings>

```
bool smtrat::cad::BaseProjection< Settings >::isPurged (
            std::size_t level,
            std::size_t id )  [inline], [protected], [inherited]
```

15.305.3.19 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.

15.305.3.20 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.

15.305.3.21 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.

15.305.3.22 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 >.

15.305.3.23 reset()

```
template<typename Settings >
void smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings
>::reset ( )  [inline]
```

Resets all datastructures, use the given variables from now on.

15.305.3.24 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.

15.305.3.25 size() [1/4] template<typename Settings >

```
std::size_t smtrat::cad::BaseProjection< Settings >::size [inline]
```

15.305.3.26 size() [2/4] template<typename Settings>

```
std::size_t smtrat::cad::BaseProjection< Settings >::size ( ) const [inline], [inherited]
```

15.305.3.27 size() [3/4] template<typename Settings >

```
std::size_t smtrat::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 smtrat::cad::BaseProjection< Settings >.

15.305.3.28 size() [4/4] template<typename Settings >

```
virtual std::size_t smtrat::cad::BaseProjection< Settings >::size
```

15.305.3.29 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.

15.305.3.30 vars() template<typename Settings>

```
const auto& smtrat::cad::BaseProjection< Settings >::vars ( ) const [inline], [inherited]
```

Returns the variables used for projection.

15.305.4 Friends And Related Function Documentation

15.305.4.1 operator<< template<typename Settings >

```
template<typename S >
std::ostream& operator<< (
            std::ostream & os,
            const ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, S > & p
) [friend]
```

15.305.5 Field Documentation

15.305.5.1 mConstraints template<typename Settings>

```
const Constraints& smtrat::cad::BaseProjection< Settings >::mConstraints [protected], [inherited]
```

15.305.5.2 mInfo `template<typename Settings>`

`ProjectionInformation smtrat::cad::BaseProjection< Settings >::mInfo [protected], [inherited]`

Additional info on projection, projection levels and projection polynomials.

15.305.5.3 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.

15.306 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 LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::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 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
- 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 _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.
- 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 >())
- 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 _preferLeft↩ Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩ ::vector< FormulaT >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩ AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩ CaseWeak=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.

15.306.1 Detailed Description

A base class for all kind of theory solving methods.

15.306.2 Member Enumeration Documentation

15.306.2.1 LemmaType

`enum smtrat::Module::LemmaType : unsigned [strong]`

Enumerator

NORMAL	
PERMANENT	

15.306.3 Constructor & Destructor Documentation

15.306.3.1 Module()

```
smtrat::Module::Module (
    const ModuleInput * _formula,
    Conditionals & _foundAnswer,
    Manager * _manager = nullptr,
    std::string module_name = "Module" )
```

Constructs a module.

Parameters

_formula	The formula passed to this module, called received formula.
_foundAnswer	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.
module_name	Name of this module as string for printing.

15.306.3.2 ~Module() `smtrat::Module::~Module ( ) [virtual]`

Destructs a module.

15.306.4 Member Function Documentation

15.306.4.1 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

_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.

15.306.4.2 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

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

15.306.4.3 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

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 smtrat::SATModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::VSModule< Settings >, smtrat::PFEModule< Settings >, smtrat::EQModule< Setting smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >,

smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Sett smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, and smtrat::GBModule< Settings >.

15.306.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.306.4.5 addLemma()

```
void smtrat::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

_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.

15.306.4.6 addOrigin()

```
void smtrat::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

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

15.306.4.7 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::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

_subformula	The sub-formula of the received formula to copy.

15.306.4.8 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

_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.

15.306.4.9 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

_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.

15.306.4.10 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

_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.

15.306.4.11 allModels() `const std::vector<Model>& smtrat::Module::allModels ( ) const [inline]`

Returns All satisfying assignments, if existent.

15.306.4.12 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.

15.306.4.13 answerFound() `const smtrat::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.

15.306.4.14 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.

15.306.4.15 branchAt() [1/6]

```
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]
```

15.306.4.16 branchAt() [2/6]

```
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]
```

15.306.4.17 branchAt() [3/6]

```
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]
```

15.306.4.18 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.306.4.19 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P
>> = carl::dummy>
bool smtrat::Module::branchAt (
    const typename P::PolyType & _poly,
```

```
            bool _integral,
            const Rational & _value,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false,
            const std::vector< FormulaT > & _premise = std::vector<FormulaT>() )  [inline],
[protected]
```

15.306.4.20 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P
>> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false )  [inline], [protected]
```

15.306.4.21 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in smtrat::PModule.

15.306.4.22 checkCore()

```
Answer smtrat::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 smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::SATModule< Settings >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRAS smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::EQPreprocessingModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::VSModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::GBModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, smtrat::LVEModule< Settings >, smtrat::ESModule< Settings >, smtrat::GBPPModule< Settings >, smtrat::CNFerModule, smtrat::BEModule< Settings >, smtrat::MCBModule< Settings >, smtrat::PFEModule< Settings >, smtrat::SplitSOSModule< Setti smtrat::EMModule< Settings >, and smtrat::SymmetryModule< Settings >.

15.306.4.23 checkInfSubsetForMinimality()

```
void smtrat::Module::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.

Parameters

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.306.4.24 checkModel()

```
unsigned smtrat::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.

15.306.4.25 cleanModel()

```
void smtrat::Module::cleanModel ( ) const [inline], [protected]
```

Substitutes variable occurrences by its model value in the model values of other variables.

15.306.4.26 clearLemmas()

```
void smtrat::Module::clearLemmas ( ) [inline]
```

Deletes all yet found lemmas.

15.306.4.27 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected]`

Clears the assignment, if any was found.

15.306.4.28 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected]`

Clears all assignments, if any was found.

15.306.4.29 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected]`

15.306.4.30 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const
```

15.306.4.31 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

_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.

15.306.4.32 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( )`

15.306.4.33 constraintsToInform()

```
const carl::FastSet<FormulaT>& smtrat::Module::constraints↩
ToInform ( ) const [inline]
```

Returns The constraints which the backends must be informed about.

15.306.4.34 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

15.306.4.35 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)

15.306.4.36 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

_constraint	The constraint to remove from internal data structures.

15.306.4.37 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.306.4.38 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const  [protected]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.306.4.39 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.306.4.40 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const
```

Excludes all variables from the current model, which do not occur in the received formula.

15.306.4.41 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
            const std::vector< FormulaT > & _origins ) const [protected]
```

Parameters

_origins	

Returns

15.306.4.42 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.

15.306.4.43 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.

15.306.4.44 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static]`

15.306.4.45 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected]`

Stores the trivial infeasible subset being the set of received formulas.

15.306.4.46 getBackendsAllModels()

`void smtrat::Module::getBackendsAllModels ( ) const [protected]`

Stores all models of a backend in the list of all models of this module.

15.306.4.47 getBackendsModel()

`void smtrat::Module::getBackendsModel ( ) const [protected]`

15.306.4.48 getInfeasibleSubsets() [1/2]

`void smtrat::Module::getInfeasibleSubsets ( ) [protected]`

Copies the infeasible subsets of the passed formula.

15.306.4.49 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.306.4.50 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.306.4.51 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.

15.306.4.52 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected]
```

Parameters

_formula	
_origins	

15.306.4.53 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected]
```

Parameters

_formula	
_origins	

15.306.4.54 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.306.4.55 getReceivedFormulaSimplified() `std::pair< bool, FormulaT > smtrat::Module::getReceivedFormulaSimplified ( ) [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 reveived formula, if the Boolean is true.

Reimplemented in smtrat::PModule.

15.306.4.56 hasLemmas() `bool smtrat::Module::hasLemmas ( ) [inline]`

Checks whether this module or any of its backends provides any lemmas.

15.306.4.57 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).

15.306.4.58 id() `std::size_t smtrat::Module::id ( ) const [inline]`

Returns A unique ID to identify this module instance.

15.306.4.59 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.

15.306.4.60 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.306.4.61 informBackends()

```
void smtrat::Module::informBackends (
    const FormulaT & _constraint ) [inline], [protected]
```

Informs all backends of this module about the given constraint.

Parameters

_constraint	The constraint to inform about.

15.306.4.62 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, and smtrat::ICPModule< ICPSettings1 >.

15.306.4.63 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline]
```

Returns The position of the first constraint of which no backend has been informed about.

15.306.4.64 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::FPPModule< Settings >.

15.306.4.65 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline]`

15.306.4.66 isLemmaLevel() `bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.306.4.67 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().

15.306.4.68 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.

15.306.4.69 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.306.4.70 model() `const Model& smtrat::Module::model ( ) const [inline]`

Returns The assignment of the current satisfiable result, if existent.

15.306.4.71 modelsDisjoint() `bool smtrat::Module::modelsDisjoint ( const Model & _modelA, const Model & _modelB ) [static], [protected]`

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.

15.306.4.72 moduleName() `virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual]`

Returns The name of the given module type as name.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::EQPreprocessingModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::VSModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::GBModule< Settings >, smtrat::IntEqModule< Setti smtrat::CubeLIAModule< Settings >, smtrat::CurryModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IncWidthModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::BEModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::SplitSOSModule< Settings >.

15.306.4.73 objective() `carl::Variable smtrat::Module::objective ( ) const [inline]`

15.306.4.74 originInReceivedFormula() `bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected]`

15.306.4.75 passedFormulaBegin() `ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected]`

Returns An iterator to the end of the passed formula. TODO: disable this method

15.306.4.76 passedFormulaEnd() `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected]`

Returns An iterator to the end of the passed formula.

15.306.4.77 pPassedFormula() `const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline]`

Returns A pointer to the formula passed to the backends of this module.

15.306.4.78 pReceivedFormula() `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline]`

Returns A pointer to the formula passed to this module.

15.306.4.79 print() `void smtrat::Module::print ( const std::string & _initiation = "***" ) const`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.306.4.80 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

_out	The stream to print the assignment on.

15.306.4.81 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.306.4.82 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

_out	The stream to print the assignment on.

15.306.4.83 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.306.4.84 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.306.4.85 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

_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.

15.306.4.86 receivedFormulaChecked()

```
void smtrat::Module::receivedFormulaChecked ( ) [inline]
```

Notifies that the received formulas has been checked.

15.306.4.87 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩
InfeasibleSubset (
    ModuleInput::const_iterator _subformula ) [inline], [protected]
```

Stores an infeasible subset consisting only of the given received formula.

15.306.4.88 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const [inline]
```

15.306.4.89 remove()

```
void smtrat::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

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

15.306.4.90 removeCore()

```
virtual void smtrat::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

formula	The sub formula of the received formula to remove.

Reimplemented in smtrat::SATModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::VSModule< Settings >, smtrat::PFEModule< Settings >, smtrat::EQModule< Setting smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Sett smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, and smtrat::GBModule< Settings >.

15.306.4.91 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin
(
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected]
```

15.306.4.92 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected]
```

15.306.4.93 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline]
```

Returns A reference to the formula passed to the backends of this module.

15.306.4.94 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline]
```

Returns A reference to the formula passed to this module.

15.306.4.95 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( ) [inline],
[protected], [virtual]
```

Reimplemented in smtrat::PModule.

15.306.4.96 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

_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 in smtrat::PModule.

15.306.4.97 setId()

```
void smtrat::Module::setId ( std::size_t _id ) [inline]
```

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.306.4.98 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

_threadPriority	The priority to set this module's thread priority to.

15.306.4.99 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.

15.306.4.100 splitUnequalConstraint()

```
void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected]
```

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.306.4.101 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline]`

Returns The priority of this module to get a thread for running its check procedure.

15.306.4.102 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 >.

15.306.4.103 updateLemmas() `void smtrat::Module::updateLemmas ( )`

Stores all lemmas of any backend of this module in its own lemma vector.

15.306.4.104 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::SATModule< Settings >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRAS smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::EQPreprocessingModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::PModule, smtrat::VSModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, smtrat::LVEModule< Settings >, smtrat::MCBModule< Settings >, smtrat::ESModule< Settings >, and smtrat::GBPPModule< Settings >.

15.306.4.105 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).

15.306.5 Field Documentation

15.306.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected]`

The backends of this module which have been used.

15.306.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected]`

Stores all satisfying assignments.

15.306.5.3 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.

15.306.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected]`

Stores the constraints which the backends must be informed about.

15.306.5.5 mFinalCheck `bool smtrat::Module::mFinalCheck [protected]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

15.306.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static]`

The beginning of the cyclic buffer storing the last branches.

15.306.5.7 mFirstSubformulaToPass `ModuleInput::iterator smtrat::Module::mFirstSubformulaTo↩Pass [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.

15.306.5.8 mFirstUncheckedReceivedSubformula `ModuleInput::const_iterator smtrat::Module::m↩FirstUncheckedReceivedSubformula [protected]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

15.306.5.9 mFoundAnswer `Conditionals smtrat::Module::mFoundAnswer [protected]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.306.5.10 mFullCheck `bool smtrat::Module::mFullCheck [protected]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.306.5.11 mInfeasibleSubsets `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected]`

Stores the infeasible subsets.

15.306.5.12 mInformedConstraints `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected]`

Stores the position of the first constraint of which no backend has been informed about.

15.306.5.13 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.

15.306.5.14 mLemmas `std::vector<Lemma> smtrat::Module::mLemmas [protected]`

Stores the lemmas being valid formulas this module or its backends made.

15.306.5.15 mModel `Model smtrat::Module::mModel [mutable], [protected]`

Stores the assignment of the current satisfiable result, if existent.

15.306.5.16 mModelComputed `bool smtrat::Module::mModelComputed [mutable], [protected]`

True, if the model has already been computed.

15.306.5.17 mNumOfBranchVarsToStore `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static]`

The number of different variables to consider for a probable infinite loop of branchings.

15.306.5.18 mObjectiveVariable `carl::Variable smtrat::Module::mObjectiveVariable [protected]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

15.306.5.19 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static]`

Reusable splitting variables.

15.306.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected]`

A reference to the manager.

15.306.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.306.5.22 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.

15.306.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected]`

15.306.5.24 mUsedBackends `std::vector<Module*> smtrat::Module::mUsedBackends [protected]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

15.306.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected]`

Maps variables to the number of their occurrences.

15.307 smtrat::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

15.307.1 Constructor & Destructor Documentation

15.307.1.1 ModuleFactory() `template<typename Module> smtrat::ModuleFactory< Module >::ModuleFactory ( ) [inline]`

15.307.2 Member Function Documentation

15.307.2.1 create()

```
template<typename Module>
Module* smtrat::ModuleFactory< Module >::create (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager *const _manager )  [inline], [virtual]
```

Implements smtrat::AbstractModuleFactory.

15.307.2.2 moduleName()

```
template<typename Module>
std::string smtrat::ModuleFactory< Module >::moduleName ( ) const  [inline], [virtual]
```

Implements smtrat::AbstractModuleFactory.

15.308 smtrat::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 isConstraintConjunction () const
- bool isConstraintLiteralConjunction () const
- bool isRealConstraintConjunction () const
- bool isRealConstraintLiteralConjunction () const
- bool isIntegerConstraintConjunction () const
- bool isIntegerConstraintLiteralConjunction () const
- bool containsBitVectorConstraints () const
- bool containsBooleanVariables () const
- bool containsRealVariables () const
- bool containsIntegerVariables () const
- bool containsUninterpretedEquations () const
- bool isOnlyPropositional () const
- unsigned satisfiedBy (const Model &_assignment) const
- unsigned satisfiedBy (const EvalRationalMap &_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 _mightBe↩Conjunction=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

15.308.1 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.

15.308.2 Member Typedef Documentation

15.308.2.1 const_iterator

`typedef super::const_iterator smtrat::ModuleInput::const_iterator`

Passing through the list::const_iterator.

15.308.2.2 iterator

`typedef super::iterator smtrat::ModuleInput::iterator`

Passing through the list::iterator.

15.308.3 Constructor & Destructor Documentation

15.308.3.1 ModuleInput()

`smtrat::ModuleInput::ModuleInput ( ) [inline]`

Constructs a module input.

15.308.4 Member Function Documentation

15.308.4.1 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 )
```

15.308.4.2 add() [2/4]

```
std::pair<iterator,bool> smtrat::ModuleInput::add (
    const FormulaT & _formula,
    bool _mightBeConjunction = true ) [inline]
```

15.308.4.3 add() [3/4]

```
std::pair<iterator,bool> smtrat::ModuleInput::add (
    const FormulaT & _formula,
    const FormulaT & _origins,
    bool _mightBeConjunction = true ) [inline]
```

15.308.4.4 add() [4/4]

```
std::pair<iterator,bool> smtrat::ModuleInput::add (
    const FormulaT & _formula,
    const std::shared_ptr< FormulasT > & _origins,
    bool _mightBeConjunction = true ) [inline]
```

15.308.4.5 addOrigin()

```
void smtrat::ModuleInput::addOrigin (
    iterator _formula,
    const FormulaT & _origin ) [inline]
```

15.308.4.6 back()

```
const auto& smtrat::ModuleInput::back ( ) const [inline]
```

15.308.4.7 begin() [1/2]

```
iterator smtrat::ModuleInput::begin ( ) [inline]
```

15.308.4.8 begin() [2/2]

```
const_iterator smtrat::ModuleInput::begin ( ) const [inline]
```

15.308.4.9 clearOrigins()

```
void smtrat::ModuleInput::clearOrigins (
    iterator _formula ) [inline]
```

15.308.4.10 contains()

```
bool smtrat::ModuleInput::contains (
    const FormulaT & _subformula ) const [inline]
```

Parameters

_subformula	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.

15.308.4.11 containsBitVectorConstraints() `bool smtrat::ModuleInput::containsBitVectorConstraints ( ) const [inline]`

Returns true, if this formula contains bit vector constraints; false, otherwise.

15.308.4.12 containsBooleanVariables() `bool smtrat::ModuleInput::containsBooleanVariables ( ) const [inline]`

Returns true, if this formula contains Boolean variables; false, otherwise.

15.308.4.13 containsIntegerVariables() `bool smtrat::ModuleInput::containsIntegerVariables ( ) const [inline]`

Returns true, if this formula contains Boolean variables; false, otherwise.

15.308.4.14 containsRealVariables() `bool smtrat::ModuleInput::containsRealVariables ( ) const [inline]`

Returns true, if this formula contains Boolean variables; false, otherwise.

15.308.4.15 containsUninterpretedEquations() `bool smtrat::ModuleInput::containsUninterpreted↩ Equations ( ) const [inline]`

Returns true, if this formula contains uninterpreted equations; false, otherwise.

15.308.4.16 empty() `bool smtrat::ModuleInput::empty ( ) const [inline]`

15.308.4.17 end() [1/2] `iterator smtrat::ModuleInput::end ( ) [inline]`

15.308.4.18 end() [2/2] `const_iterator smtrat::ModuleInput::end ( ) const [inline]`

15.308.4.19 erase() `ModuleInput::iterator smtrat::ModuleInput::erase ( iterator _formula )`

15.308.4.20 find() [1/4]

```
ModuleInput::iterator smtrat::ModuleInput::find (
    const FormulaT & _formula )
```

15.308.4.21 find() [2/4]

```
ModuleInput::const_iterator smtrat::ModuleInput::find (
    const FormulaT & _formula ) const
```

15.308.4.22 find() [3/4]

```
ModuleInput::iterator smtrat::ModuleInput::find (
    const_iterator _hint,
    const FormulaT & _formula )
```

15.308.4.23 find() [4/4]

```
ModuleInput::const_iterator smtrat::ModuleInput::find (
    const_iterator _hint,
    const FormulaT & _formula ) const
```

15.308.4.24 gatherVariables()

```
void smtrat::ModuleInput::gatherVariables (
    carl::carlVariables & vars ) const [inline]
```

15.308.4.25 isConstraintConjunction()

```
bool smtrat::ModuleInput::isConstraintConjunction ( )
const [inline]
```

Returns true, if this formula is a conjunction of constraints; false, otherwise.

15.308.4.26 isConstraintLiteralConjunction()

```
bool smtrat::ModuleInput::isConstraintLiteral↩
Conjunction ( ) const [inline]
```

Returns true, if this formula is a conjunction of literals of constraints; false, otherwise.

15.308.4.27 isIntegerConstraintConjunction()

```
bool smtrat::ModuleInput::isIntegerConstraint↩
Conjunction ( ) const [inline]
```

Returns true, if this formula is a conjunction of integer constraints; false, otherwise.

15.308.4.28 isIntegerConstraintLiteralConjunction()

```
bool smtrat::ModuleInput::isIntegerConstraint↩
LiteralConjunction ( ) const [inline]
```

Returns true, if this formula is a conjunction of literals of integer constraints; false, otherwise.

15.308.4.29 isOnlyPropositional() `bool smtrat::ModuleInput::isOnlyPropositional ( ) const [inline]`

Returns true, if this formula is propositional; false, otherwise.

15.308.4.30 isRealConstraintConjunction() `bool smtrat::ModuleInput::isRealConstraintConjunction ( ) const [inline]`

Returns true, if this formula is a conjunction of real constraints; false, otherwise.

15.308.4.31 isRealConstraintLiteralConjunction() `bool smtrat::ModuleInput::isRealConstraint↩ LiteralConjunction ( ) const [inline]`

Returns true, if this formula is a conjunction of literals of real constraints; false, otherwise.

15.308.4.32 operator FormulaT() `smtrat::ModuleInput::operator FormulaT ( ) const [inline], [explicit]`

15.308.4.33 properties() `carl::Condition smtrat::ModuleInput::properties ( ) const [inline]`

Returns All known properties of the underlying formula of this module input.

15.308.4.34 rbegin() `auto smtrat::ModuleInput::rbegin ( ) const [inline]`

15.308.4.35 removeOrigin()

```
bool smtrat::ModuleInput::removeOrigin (
    iterator _formula,
    const FormulaT & _origin )
```

15.308.4.36 removeOrigins()

```
bool smtrat::ModuleInput::removeOrigins (
    iterator _formula,
    const std::shared_ptr< FormulasT > & _origins )
```

15.308.4.37 rend() `auto smtrat::ModuleInput::rend ( ) const [inline]`

15.308.4.38 satisfiedBy() [1/2]

```
unsigned smtrat::ModuleInput::satisfiedBy (
    const EvalRationalMap & _assignment ) const
```

Parameters

_assignment	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.

15.308.4.39 satisfiedBy() [2/2]

```
unsigned smtrat::ModuleInput::satisfiedBy (
    const Model & _assignment ) const
```

Parameters

_assignment	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.

15.308.4.40 size()

```
size_t smtrat::ModuleInput::size (
    void ) const [inline]
```

15.308.4.41 updateProperties()

```
void smtrat::ModuleInput::updateProperties ( )
```

Updates all properties of the formula underlying this module input.

15.308.5 Friends And Related Function Documentation

15.308.5.1 Manager

```
friend class Manager [friend]
```

15.308.5.2 Module

```
friend class Module [friend]
```

15.309 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

15.309.1 Constructor & Destructor Documentation

15.309.1.1 ModuleSettings() `smtrat::settings::ModuleSettings::ModuleSettings ( ) [inline]`

15.309.2 Member Function Documentation

15.309.2.1 get()

```
template<typename T >
T smtrat::settings::ModuleSettings::get (
            const std::string & key,
            T default_value ) const [inline]
```

15.309.2.2 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]
```

15.309.3 Field Documentation

15.309.3.1 parameters `std::vector<std::string> smtrat::settings::ModuleSettings::parameters`

15.310 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 ()

15.310.1 Member Function Documentation

15.310.1.1 continue_all() `void smtrat::Module::ModuleStatistics::continue_all ( ) [inline]`

15.310.1.2 pause_all() `void smtrat::Module::ModuleStatistics::pause_all ( ) [inline]`

15.310.1.3 start_add() `void smtrat::Module::ModuleStatistics::start_add ( ) [inline]`

15.310.1.4 start_check() `void smtrat::Module::ModuleStatistics::start_check ( ) [inline]`

15.310.1.5 start_remove() `void smtrat::Module::ModuleStatistics::start_remove ( ) [inline]`

15.310.1.6 stop_add() `void smtrat::Module::ModuleStatistics::stop_add ( ) [inline]`

15.310.1.7 stop_check() `void smtrat::Module::ModuleStatistics::stop_check ( ) [inline]`

15.310.1.8 stop_remove() `void smtrat::Module::ModuleStatistics::stop_remove ( ) [inline]`

15.311 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 ()

15.311.1 Constructor & Destructor Documentation

15.311.1.1 ModuleWrapper()

```
template<typename M>
smtrat::ModuleWrapper< M >::ModuleWrapper ( ) [inline]
```

15.311.2 Member Function Documentation

15.311.2.1 add()

```
template<typename M>
bool smtrat::ModuleWrapper< M >::add (
            const FormulaT & formula ) [inline]
```

15.311.2.2 check()

```
template<typename M>
bool smtrat::ModuleWrapper< M >::check ( ) [inline]
```

15.311.2.3 get()

```
template<typename M>
M& smtrat::ModuleWrapper< M >::get ( ) [inline], [noexcept]
```

15.311.2.4 infeasibleSubsets()

```
template<typename M>
const std::vector<FormulaSetT>& smtrat::ModuleWrapper< M >::infeasibleSubsets ( ) [inline]
```

15.311.2.5 isAsserted()

```
template<typename M>
bool smtrat::ModuleWrapper< M >::isAsserted (
            const FormulaT & formula ) [inline]
```

15.311.2.6 remove()

```
template<typename M>
void smtrat::ModuleWrapper< M >::remove (
            const FormulaT & formula ) [inline]
```

15.312 smtrat::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 >

15.312.1 Member Function Documentation

15.312.1.1 getMMonomialMapping()

```
const MonomialMap& smtrat::MonomialMappingByVariablePool↩
::getMMonomialMapping ( ) const [inline]
```

15.312.1.2 insertMonomialMapping()

```
void smtrat::MonomialMappingByVariablePool::insertMonomial↩
Mapping (
            carl::Variable variable,
            carl::Monomial::Arg monomial ) [inline]
```

15.312.1.3 isNull()

```
bool smtrat::MonomialMappingByVariablePool::isNull (
            carl::Variable variable ) [inline]
```

15.312.1.4 monomial()

```
carl::Monomial::Arg smtrat::MonomialMappingByVariablePool::monomial (
            carl::Variable variable ) [inline]
```

15.312.1.5 variable()

```
carl::Variable smtrat::MonomialMappingByVariablePool::variable (
            carl::Monomial::Arg monomial ) [inline]
```

15.312.2 Friends And Related Function Documentation

15.312.2.1 carl::Singleton< MonomialMappingByVariablePool >

```
friend class carl::Singleton<
MonomialMappingByVariablePool > [friend]
```

15.313 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

15.313.1 Constructor & Destructor Documentation

15.313.1.1 NaryExpression() [1/2]

```
smtrat::expression::NaryExpression::NaryExpression (
    NaryType _type,
    Expressions && _expressions ) [inline]
```

15.313.1.2 NaryExpression() [2/2]

```
smtrat::expression::NaryExpression::NaryExpression (
    NaryType _type,
    const std::initializer_list< Expression > & _expressions ) [inline]
```

15.313.2 Member Function Documentation

15.313.2.1 normalize()

```
void smtrat::expression::NaryExpression::normalize ( ) [inline]
```

15.313.3 Field Documentation

15.313.3.1 expressions

```
Expressions smtrat::expression::NaryExpression::expressions
```

15.313.3.2 type

```
NaryType smtrat::expression::NaryExpression::type
```

15.314 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

15.314.1 Member Function Documentation

15.314.1.1 operator()() [1/7]

```
const ExpressionContent* smtrat::expression::simplifier::Base↩
Simplifier::operator() (
    const BinaryExpression & expr ) const [inline], [inherited]
```

15.314.1.2 operator()() [2/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const carl::Variable & expr ) const  [inline], [inherited]
```

15.314.1.3 operator()() [3/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const ExpressionContent * _ec ) const  [inline], [inherited]
```

15.314.1.4 operator()() [4/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const ITEExpression & expr ) const  [inline], [inherited]
```

15.314.1.5 operator()() [5/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const NaryExpression & expr ) const  [inline], [inherited]
```

15.314.1.6 operator()() [6/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const QuantifierExpression & expr ) const  [inline], [inherited]
```

15.314.1.7 operator()() [7/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const UnaryExpression & expr ) const  [inline], [inherited]
```

15.314.1.8 simplify() [1/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const BinaryExpression &  ) const  [inline], [protected], [virtual], [inherited]
```

15.314.1.9 simplify() [2/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const carl::Variable &  ) const  [inline], [protected], [virtual], [inherited]
```

15.314.1.10 simplify() [3/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const ITEExpression &  ) const  [inline], [protected], [virtual], [inherited]
```

15.314.1.11 simplify() [4/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const NaryExpression &  ) const  [inline], [protected], [virtual], [inherited]
```

Reimplemented in smtrat::expression::simplifier::DuplicateSimplifier, smtrat::expression::simplifier::MergeSimplifier, and smtrat::expression::simplifier::SingletonSimplifier.

15.314.1.12 simplify() [5/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::↩
BaseSimplifier::simplify (
            const QuantifierExpression &  ) const  [inline], [protected], [virtual], [inherited]
```

15.314.1.13 simplify() [6/6]

```
const ExpressionContent* smtrat::expression::simplifier::Negation↩
Simplifier::simplify (
            const UnaryExpression & expr ) const  [inline], [virtual]
```

Reimplemented from smtrat::expression::simplifier::BaseSimplifier.

15.315 smtrat::NewCADModule< Settings > Class Template Reference

```
#include <NewCADModule.h>
```

Public Types

- typedef Settings SettingsType
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::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 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 hasValidInfeasibleSubset () 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 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 &_branching↩Value) 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 _preferLeft↩Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩::vector< FormulaT >())

- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩CaseWeak=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.

15.315.1 Member Typedef Documentation

15.315.1.1 SettingsType `template<typename Settings >`

```
typedef Settings smtrat::NewCADModule< Settings >::SettingsType
```

15.315.2 Member Enumeration Documentation

15.315.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

15.315.3 Constructor & Destructor Documentation

15.315.3.1 NewCADModule() `template<class Settings >`

```
smtrat::NewCADModule< Settings >::NewCADModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

15.315.3.2 ~NewCADModule() `template<class Settings >`

```
smtrat::NewCADModule< Settings >::~NewCADModule ( )
```

15.315.4 Member Function Documentation

15.315.4.1 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.

15.315.4.2 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 >.

15.315.4.3 addCore()

```
template<class 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

_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.

15.315.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.315.4.5 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.

15.315.4.6 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.

15.315.4.7 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.

15.315.4.8 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.

15.315.4.9 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.

15.315.4.10 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

_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.

15.315.4.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const  [inline],
[inherited]
```

Returns All satisfying assignments, if existent.

15.315.4.12 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.

15.315.4.13 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.

15.315.4.14 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.

15.315.4.15 branchAt() [1/6]

```
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]
```

15.315.4.16 branchAt() [2/6]

```
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]
```

15.315.4.17 branchAt() [3/6]

```
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]
```

15.315.4.18 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.315.4.19 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P
>> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            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]
```

15.315.4.20 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P
>> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

15.315.4.21 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in smtrat::PModule.

15.315.4.22 checkCore()

```
template<class Settings >
Answer smtrat::NewCADModule< Settings >::checkCore ( ) [virtual]
```

Checks the received formula for consistency.

Parameters

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

15.315.4.23 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
    std::vector< FormulaSetT >::const_iterator _infsubset,
    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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.315.4.24 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.

15.315.4.25 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.315.4.26 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.315.4.27 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.315.4.28 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.315.4.29 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.315.4.30 collectOrigins() [1/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

15.315.4.31 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

_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.

15.315.4.32 collectTheoryPropagations()

`void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.315.4.33 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ToInform ( ) const [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.315.4.34 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

15.315.4.35 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)

15.315.4.36 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

_constraint	The constraint to remove from internal data structures.

15.315.4.37 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.315.4.38 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.315.4.39 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.315.4.40 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.315.4.41 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.315.4.42 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.

15.315.4.43 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.

15.315.4.44 freeSplittingVariable()

```
static void smtrat::Module::freeSplittingVariable (
    const FormulaT & _splittingVariable ) [inline], [static], [inherited]
```

15.315.4.45 generateTrivialInfeasibleSubset()

```
void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]
```

Stores the trivial infeasible subset being the set of received formulas.

15.315.4.46 getBackendsAllModels()

```
void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]
```

Stores all models of a backend in the list of all models of this module.

15.315.4.47 getBackendsModel()

```
void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]
```

15.315.4.48 getInfeasibleSubsets() [1/2]

```
void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]
```

Copies the infeasible subsets of the passed formula.

15.315.4.49 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.315.4.50 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.315.4.51 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.

15.315.4.52 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.315.4.53 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.315.4.54 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.315.4.55 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 reveived formula, if the Boolean is true.

Reimplemented in smtrat::PModule.

15.315.4.56 hasLemmas()

`bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

15.315.4.57 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).

15.315.4.58 id()

`std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns A unique ID to identify this module instance.

15.315.4.59 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.

15.315.4.60 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.315.4.61 informBackends()

```
void smtrat::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.

15.315.4.62 informCore()

```
template<class 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

_constraint	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.

15.315.4.63 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.315.4.64 init()

```
template<class 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.

15.315.4.65 is_minimizing()

`bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.315.4.66 isLemmaLevel()

`bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.315.4.67 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().

15.315.4.68 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.

15.315.4.69 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.315.4.70 model()

`const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.315.4.71 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

_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.

15.315.4.72 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.

15.315.4.73 objective()

```
carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]
```

15.315.4.74 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
    const FormulaT & _origin ) const [protected], [inherited]
```

15.315.4.75 passedFormulaBegin()

```
ModuleInput::iterator smtrat::Module::passedFormulaBegin ( )
[inline], [protected], [inherited]
```

Returns An iterator to the end of the passed formula. TODO: disable this method

15.315.4.76 passedFormulaEnd()

```
ModuleInput::iterator smtrat::Module::passedFormulaEnd ( )
[inline], [protected], [inherited]
```

Returns An iterator to the end of the passed formula.

15.315.4.77 pPassedFormula()

```
const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline],
[inherited]
```

Returns A pointer to the formula passed to the backends of this module.

15.315.4.78 pReceivedFormula()

```
const ModuleInput* smtrat::Module::pReceivedFormula ( ) const
[inline], [inherited]
```

Returns A pointer to the formula passed to this module.

15.315.4.79 print()

```
void smtrat::Module::print (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.315.4.80 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

_out	The stream to print the assignment on.

15.315.4.81 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.315.4.82 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

_out	The stream to print the assignment on.

15.315.4.83 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.315.4.84 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.315.4.85 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.

15.315.4.86 receivedFormulaChecked()

```
void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]
```

Notifies that the received formulas has been checked.

15.315.4.87 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩InfeasibleSubset (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

15.315.4.88 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const [inline], [inherited]
```

15.315.4.89 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

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

15.315.4.90 removeCore()

```
template<class 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.

15.315.4.91 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin
(
            ModuleInput::iterator _formula,
            const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.315.4.92 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

15.315.4.93 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline],
[inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.315.4.94 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.315.4.95 runBackends() [1/2] virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]

Reimplemented in smtrat::PModule.

15.315.4.96 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

_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 in smtrat::PModule.

15.315.4.97 setId() void smtrat::Module::setId (std::size_t _id) [inline], [inherited]

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.315.4.98 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.

15.315.4.99 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.

15.315.4.100 splitUnequalConstraint()

`void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.315.4.101 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.315.4.102 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 >.

15.315.4.103 updateLemmas()

`void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.315.4.104 updateModel()

`template<class 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.

15.315.4.105 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).

15.315.5 Field Documentation

15.315.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.315.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.315.5.3 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.

15.315.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.315.5.5 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

15.315.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.315.5.7 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.

15.315.5.8 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.

15.315.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.315.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.315.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.315.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.315.5.13 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.

15.315.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.315.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.315.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.315.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.315.5.18 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.

15.315.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]`

Reusable splitting variables.

15.315.5.20 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.315.5.21 mSmallerMusesCheckCounter

`unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.315.5.22 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.

15.315.5.23 mTheoryPropagations

`std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]`

15.315.5.24 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).

15.315.5.25 mVariableCounters

`std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.316 smtrat::NewCoveringModule< Settings > Class Template Reference

```
#include <NewCoveringModule.h>
```

Public Types

- using SettingsType = Settings
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::PERMANENT = 1 }

Public Member Functions

- NewCoveringModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=nullptr)
- ~NewCoveringModule ()
- 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.
- size_t addConstraintsSAT ()

  Adds the given constraint to the backend.
- void removeConstraintsSAT ()

  Removes the given constraints from the backend.
- void addConstraintsUNSAT ()

  Adds the given constraint to the backend.
- bool removeConstraintsUNSAT ()

  Removes the given constraints from the backend Also removes all stored information about the removed constraints.
- 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
- 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 _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 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 &_branching↩Value) 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 _preferLeft↩Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩::vector< FormulaT >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩AsPremise=false)

- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, 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.

15.316.1 Member Typedef Documentation

15.316.1.1 SettingsType

```
template<typename Settings >
using smtrat::NewCoveringModule< Settings >::SettingsType = Settings
```

15.316.2 Member Enumeration Documentation

15.316.2.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

15.316.3 Constructor & Destructor Documentation

15.316.3.1 NewCoveringModule()

```
template<class Settings >
smtrat::NewCoveringModule< Settings >::NewCoveringModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

15.316.3.2 ~NewCoveringModule()

```
template<class Settings >
smtrat::NewCoveringModule< Settings >::~NewCoveringModule ( )
```

15.316.4 Member Function Documentation

15.316.4.1 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.

15.316.4.2 addConstraintsSAT()

```
template<class 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

15.316.4.3 addConstraintsUNSAT()

```
template<class Settings >
void smtrat::NewCoveringModule< Settings >::addConstraintsUNSAT ( )
```

Adds the given constraint to the backend.

15.316.4.4 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

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

15.316.4.5 addCore()

```
template<class 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

_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.

15.316.4.6 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.316.4.7 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

_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.

15.316.4.8 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

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

15.316.4.9 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

_subformula	The sub-formula of the received formula to copy.

15.316.4.10 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

_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.

15.316.4.11 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

_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.

15.316.4.12 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

_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.

15.316.4.13 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline], [inherited]
```

Returns All satisfying assignments, if existent.

15.316.4.14 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.

15.316.4.15 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.

15.316.4.16 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.

15.316.4.17 branchAt() [1/6]

```
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]
```

15.316.4.18 branchAt() [2/6]

```
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]
```

15.316.4.19 branchAt() [3/6]

```
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]
```

15.316.4.20 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.316.4.21 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
    const typename P::PolyType & _poly,
    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]
```

15.316.4.22 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false )  [inline], [protected], [inherited]
```

15.316.4.23 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in smtrat::PModule.

15.316.4.24 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.

15.316.4.25 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
            std::vector< FormulaSetT >::const_iterator _infsubset,
            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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.316.4.26 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.

15.316.4.27 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.316.4.28 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.316.4.29 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.316.4.30 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.316.4.31 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.316.4.32 collectOrigins() [1/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

15.316.4.33 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.

15.316.4.34 collectTheoryPropagations()

`void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.316.4.35 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.316.4.36 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::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.316.4.37 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)

15.316.4.38 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

_constraint	The constraint to remove from internal data structures.

15.316.4.39 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.316.4.40 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.316.4.41 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.316.4.42 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceived↩
VariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.316.4.43 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.316.4.44 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.

15.316.4.45 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.

15.316.4.46 freeSplittingVariable()

```
static void smtrat::Module::freeSplittingVariable (
    const FormulaT & _splittingVariable ) [inline], [static], [inherited]
```

15.316.4.47 generateTrivialInfeasibleSubset()

```
void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]
```

Stores the trivial infeasible subset being the set of received formulas.

15.316.4.48 getBackendsAllModels()

```
void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]
```

Stores all models of a backend in the list of all models of this module.

15.316.4.49 getBackendsModel()

```
void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]
```

15.316.4.50 getInfeasibleSubsets() [1/2]

```
void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]
```

Copies the infeasible subsets of the passed formula.

15.316.4.51 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.316.4.52 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.316.4.53 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.

15.316.4.54 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.316.4.55 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.316.4.56 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.316.4.57 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 reveived formula, if the Boolean is true.

Reimplemented in smtrat::PModule.

15.316.4.58 hasLemmas() bool smtrat::Module::hasLemmas ( ) [inline], [inherited]

Checks whether this module or any of its backends provides any lemmas.

15.316.4.59 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).

15.316.4.60 id() std::size_t smtrat::Module::id ( ) const [inline], [inherited]

Returns A unique ID to identify this module instance.

15.316.4.61 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.

15.316.4.62 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.316.4.63 informBackends()

```
void smtrat::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.

15.316.4.64 informCore()

```
template<class Settings >
bool smtrat::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

_constraint	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.

15.316.4.65 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.316.4.66 init()

```
template<class Settings >
void smtrat::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 smtrat::Module.

15.316.4.67 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.316.4.68 isLemmaLevel() `bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.316.4.69 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().

15.316.4.70 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.

15.316.4.71 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.316.4.72 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.316.4.73 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

_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.

15.316.4.74 moduleName()

```
virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]
```

Returns The name of the given module type as name.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::EQPreprocessingModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::VSModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::GBModule< Settings >, smtrat::IntEqModule< Setti smtrat::CubeLIAModule< Settings >, smtrat::CurryModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IncWidthModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::BEModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::SplitSOSModule< Settings >.

15.316.4.75 objective()

```
carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]
```

15.316.4.76 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
    const FormulaT & _origin ) const [protected], [inherited]
```

15.316.4.77 passedFormulaBegin()

```
ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]
```

Returns An iterator to the end of the passed formula. TODO: disable this method

15.316.4.78 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula.

15.316.4.79 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to the backends of this module.

15.316.4.80 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to this module.

15.316.4.81 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.316.4.82 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

_out	The stream to print the assignment on.

15.316.4.83 printInfeasibleSubsets()

`void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.316.4.84 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

_out	The stream to print the assignment on.

15.316.4.85 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.316.4.86 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.316.4.87 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

_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.

15.316.4.88 receivedFormulaChecked() `void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

15.316.4.89 receivedFormulasAsInfeasibleSubset() `void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

15.316.4.90 receivedVariable() `bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

15.316.4.91 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.

15.316.4.92 removeConstraintsSAT() `template<class Settings > void smtrat::NewCoveringModule< Settings >::removeConstraintsSAT ( )`

Removes the given constraints from the backend.

Note This also removes all stored information about the removed constraints

15.316.4.93 removeConstraintsUNSAT() `template<class 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

15.316.4.94 removeCore()

```
template<class 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

_subformula	The position of the subformula to remove.

Reimplemented from smtrat::Module.

15.316.4.95 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin
(
            ModuleInput::iterator _formula,
            const FormulaT & _origin )  [inline], [protected], [inherited]
```

15.316.4.96 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins )  [inline], [protected],
[inherited]
```

15.316.4.97 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const  [inline],
[inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.316.4.98 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.316.4.99 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( )  [inline],
[protected], [virtual], [inherited]
```

Reimplemented in smtrat::PModule.

15.316.4.100 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

_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 in smtrat::PModule.

15.316.4.101 setId() `void smtrat::Module::setId ( std::size_t _id ) [inline], [inherited]`

Sets this modules unique ID to identify itself.

Parameters

↩ ↩ *id*	The id to set this module's id to.

15.316.4.102 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

_threadPriority	The priority to set this module's thread priority to.

15.316.4.103 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.

15.316.4.104 splitUnequalConstraint() `void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.316.4.105 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.316.4.106 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 >.

15.316.4.107 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.316.4.108 updateModel() `template<class 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.

15.316.4.109 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).

15.316.5 Field Documentation

15.316.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.316.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.316.5.3 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.

15.316.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.316.5.5 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

15.316.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.316.5.7 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.

15.316.5.8 mFirstUncheckedReceivedSubformula `ModuleInput::const_iterator smtrat::Module::m↩FirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

15.316.5.9 mFoundAnswer `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.316.5.10 mFullCheck `bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.316.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.316.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.316.5.13 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.

15.316.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.316.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.316.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.316.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.316.5.18 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.

15.316.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplitting↩Variables [static], [inherited]`

Reusable splitting variables.

15.316.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.316.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.316.5.22 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.

15.316.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

15.316.5.24 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).

15.316.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.317 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 &&false↩ Case)
- FormulaT rewrite_iff (const FormulaT &, FormulaSetT &&subformulas)
- FormulaT rewrite_xor (const FormulaT &, FormulasMultiT &&subformulas)

15.317.1 Member Function Documentation

15.317.1.1 rewrite_iff()

```
FormulaT smtrat::NNFPreparation::rewrite_iff (
    const FormulaT & ,
    FormulaSetT && subformulas ) [inline]
```

15.317.1.2 rewrite_implication()

```
FormulaT smtrat::NNFPreparation::rewrite_implication (
    const FormulaT & ,
    FormulaT && premise,
    FormulaT && conclusion ) [inline]
```

15.317.1.3 rewrite_ite()

```
FormulaT smtrat::NNFPreparation::rewrite_ite (
    const FormulaT & ,
    FormulaT && condition,
    FormulaT && trueCase,
    FormulaT && falseCase ) [inline]
```

15.317.1.4 rewrite_xor()

```
FormulaT smtrat::NNFPreparation::rewrite_xor (
    const FormulaT & ,
    FormulasMultiT && subformulas ) [inline]
```

15.318 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)

15.318.1 Constructor & Destructor Documentation

15.318.1.1 NNFRewriter()

```
smtrat::NNFRewriter::NNFRewriter ( ) [inline]
```

15.318.2 Member Function Documentation

15.318.2.1 enter_not()

```
void smtrat::NNFRewriter::enter_not (
    const FormulaT & ) [inline]
```

15.318.2.2 enter_xor()

```
void smtrat::NNFRewriter::enter_xor (
    const FormulaT & formula ) [inline]
```

15.318.2.3 rewrite_and()

```
FormulaT smtrat::NNFRewriter::rewrite_and (
    const FormulaT & formula,
    FormulaSetT && subformulas ) [inline]
```

15.318.2.4 rewrite_leaf()

```
FormulaT smtrat::NNFRewriter::rewrite_leaf (
    const FormulaT & formula ) [inline]
```

15.318.2.5 rewrite_not()

```
FormulaT smtrat::NNFRewriter::rewrite_not (
    const FormulaT & formula,
    FormulaT && subformula ) [inline]
```

15.318.2.6 rewrite_or()

```
FormulaT smtrat::NNFRewriter::rewrite_or (
    const FormulaT & formula,
    FormulaSetT && subformulas ) [inline]
```

15.318.2.7 rewrite_quantified()

```
FormulaT smtrat::NNFRewriter::rewrite_quantified (
    const FormulaT & formula,
    FormulaT && subformula ) [inline]
```

15.318.2.8 rewrite_ueq()

```
FormulaT smtrat::NNFRewriter::rewrite_ueq (
    const FormulaT & formula ) [inline]
```

15.318.2.9 rewrite_xor()

```
FormulaT smtrat::NNFRewriter::rewrite_xor (
    const FormulaT & formula,
    FormulasMultiT && subformulas ) [inline]
```

15.319 benchmax::ssh::Node Struct Reference

Specification of a compuation 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)

15.319.1 Detailed Description

Specification of a compuation node for the SSH backend.

15.319.2 Field Documentation

15.319.2.1 connections `std::size_t benchmax::ssh::Node::connections`

Number of concurrent connections (default is 1)

15.319.2.2 cores `unsigned long benchmax::ssh::Node::cores`

Number of cores we use per connection (default is 1)

15.319.2.3 hostname `std::string benchmax::ssh::Node::hostname`

Hostname to connect to.

15.319.2.4 password `std::string benchmax::ssh::Node::password`

Password (only used if public key authentication fails).

15.319.2.5 port `int benchmax::ssh::Node::port`

Port (default is 22)

15.319.2.6 username `std::string benchmax::ssh::Node::username`

Username.

15.320 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

15.320.1 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.

15.320.2 Constructor & Destructor Documentation

15.320.2.1 Node() [1/6]

```
delta::Node::Node ( ) [inline], [explicit]
```

Create an empty node.

15.320.2.2 Node() [2/6]

```
delta::Node::Node (
    const std::string & name,
    bool brackets = true ) [inline], [explicit]
```

Create a node with a name.

Parameters

name	Name of the node.
brackets	If not is contained in brackets.

15.320.2.3 Node() [3/6]

```
delta::Node::Node (
    const std::tuple< std::string, bool > & data ) [inline], [explicit]
```

Create a node with a name.

Parameters

data	Tuple containing the name and the brackets flag.

15.320.2.4 Node() [4/6]

```
delta::Node::Node (
    const std::tuple< std::vector< Node >, bool > & data ) [inline], [explicit]
```

Create a node with children.

Parameters

data	Tuple containing the children and the brackets flag.

15.320.2.5 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

data	Tuple containing the name, the children and the brackets flag.

15.320.2.6 Node() [6/6]

```
delta::Node::Node (
    const std::string & name,
    const std::initializer_list< Node > & children,
    bool brackets = true ) [inline], [explicit]
```

15.320.3 Member Function Documentation

15.320.3.1 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

from	Node to replace.
to	Node to replace with.

Returns Cloned node.

15.320.3.2 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 enountered, replace it with `to`.

Parameters

from	Node name to replace.
to	Node to replace with.

Returns Cloned node.

15.320.3.3 collectNames()

```
void delta::Node::collectNames (
    std::set< std::string > & names ) const [inline]
```

15.320.3.4 complexity()

```
std::size_t delta::Node::complexity ( ) const [inline]
```

Calculates the number of nodes.

Returns Number of nodes.

15.320.3.5 eliminateDefineFuns()

```
void delta::Node::eliminateDefineFuns ( ) [inline]
```

15.320.3.6 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.

15.320.3.7 recalculateSize()

```
void delta::Node::recalculateSize ( ) [inline]
```

15.320.3.8 repr()

```
std::string delta::Node::repr (
    bool longRepr = false ) const [inline]
```

Returns a string representation of this node.

Returns String of the node.

15.320.4 Field Documentation

15.320.4.1 brackets `bool delta::Node::brackets`

Flag if the node was contained in brackets.

15.320.4.2 children `std::vector<Node> delta::Node::children`

Children of the node.

15.320.4.3 name `std::string delta::Node::name`

Name of the node.

15.320.4.4 size `std::size_t delta::Node::size`

15.321 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

15.321.1 Constructor & Destructor Documentation

15.321.1.1 NodePrinter() `template<bool pretty>`

```
delta::NodePrinter< pretty >::NodePrinter (
            const Node & n )  [inline]
```

15.321.2 Member Function Documentation

15.321.2.1 pretty_print() `template<bool pretty>`

```
void delta::NodePrinter< pretty >::pretty_print (
            std::ostream & os,
            const Node & n,
            std::string indent = "" ) const  [inline]
```

15.321.2.2 print() `template<bool pretty>`

```
void delta::NodePrinter< pretty >::print (
            std::ostream & os,
            const Node & n ) const  [inline]
```

15.321.3 Field Documentation

15.321.3.1 node `template<bool pretty>`

```
const Node& delta::NodePrinter< pretty >::node
```

15.322 smtrat::mcsat::onecellcad::recursive::NoHeuristic Struct Reference

`#include <Explanation.h>`

Static Public Attributes

- static constexpr int heuristic = 0

15.322.1 Field Documentation

15.322.1.1 heuristic `constexpr int smtrat::mcsat::onecellcad::recursive::NoHeuristic::heuristic = 0 [static], [constexpr]`

15.323 smtrat::NRAILModule< Settings > Class Template Reference

`#include <NRAILModule.h>`

Public Types

- typedef Settings SettingsType
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::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

  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 hasValidInfeasibleSubset () 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 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 &_branching↩Value) 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 _preferLeft↩Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩::vector< FormulaT >())

- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩CaseWeak=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.

15.323.1 Member Typedef Documentation

15.323.1.1 SettingsType

```
template<typename Settings >
typedef Settings smtrat::NRAILModule< Settings >::SettingsType
```

15.323.2 Member Enumeration Documentation

15.323.2.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

15.323.3 Constructor & Destructor Documentation

15.323.3.1 NRAILModule()

```
template<class Settings >
smtrat::NRAILModule< Settings >::NRAILModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

15.323.3.2 ~NRAILModule()

```
template<class Settings >
smtrat::NRAILModule< Settings >::~NRAILModule ( )
```

15.323.4 Member Function Documentation

15.323.4.1 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.

15.323.4.2 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 >.

15.323.4.3 addCore()

```
template<class 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.

15.323.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.323.4.5 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.

15.323.4.6 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.

15.323.4.7 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.

15.323.4.8 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.

15.323.4.9 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.

15.323.4.10 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

_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.

15.323.4.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const  [inline],
[inherited]
```

Returns All satisfying assignments, if existent.

15.323.4.12 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.

15.323.4.13 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.

15.323.4.14 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.

15.323.4.15 branchAt() [1/6]

```
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]
```

15.323.4.16 branchAt() [2/6]

```
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]
```

15.323.4.17 branchAt() [3/6]

```
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]
```

15.323.4.18 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.323.4.19 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
    const typename P::PolyType & _poly,
    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]
```

15.323.4.20 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
    const typename P::PolyType & _poly,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

15.323.4.21 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in smtrat::PModule.

15.323.4.22 checkCore()

```
template<class 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 smtrat::Module.

15.323.4.23 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
    std::vector< FormulaSetT >::const_iterator _infsubset,
    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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.323.4.24 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.

15.323.4.25 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.323.4.26 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.323.4.27 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.323.4.28 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.323.4.29 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.323.4.30 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

15.323.4.31 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

_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.

15.323.4.32 collectTheoryPropagations()

`void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.323.4.33 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ToInform ( ) const [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.323.4.34 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

15.323.4.35 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)

15.323.4.36 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

_constraint	The constraint to remove from internal data structures.

15.323.4.37 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.323.4.38 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.323.4.39 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.323.4.40 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceived↩
VariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.323.4.41 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.323.4.42 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.

15.323.4.43 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.

15.323.4.44 freeSplittingVariable()

```
static void smtrat::Module::freeSplittingVariable (
    const FormulaT & _splittingVariable ) [inline], [static], [inherited]
```

15.323.4.45 generateTrivialInfeasibleSubset()

```
void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]
```

Stores the trivial infeasible subset being the set of received formulas.

15.323.4.46 getBackendsAllModels()

```
void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]
```

Stores all models of a backend in the list of all models of this module.

15.323.4.47 getBackendsModel()

```
void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]
```

15.323.4.48 getInfeasibleSubsets() [1/2]

```
void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]
```

Copies the infeasible subsets of the passed formula.

15.323.4.49 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.323.4.50 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.323.4.51 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.

15.323.4.52 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.323.4.53 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.323.4.54 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.323.4.55 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 reveived formula, if the Boolean is true.

Reimplemented in smtrat::PModule.

15.323.4.56 hasLemmas() bool smtrat::Module::hasLemmas ( ) [inline], [inherited]

Checks whether this module or any of its backends provides any lemmas.

15.323.4.57 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).

15.323.4.58 id() std::size_t smtrat::Module::id ( ) const [inline], [inherited]

Returns A unique ID to identify this module instance.

15.323.4.59 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.

15.323.4.60 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.323.4.61 informBackends()

```
void smtrat::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.

15.323.4.62 informCore()

```
template<class Settings >
bool smtrat::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

_constraint	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.

15.323.4.63 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.323.4.64 init()

```
template<class Settings >
void smtrat::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 smtrat::Module.

15.323.4.65 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.323.4.66 isLemmaLevel() `bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.323.4.67 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().

15.323.4.68 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.

15.323.4.69 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.323.4.70 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.323.4.71 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

_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.

15.323.4.72 moduleName()

```
template<typename Settings >
std::string smtrat::NRAILModule< Settings >::moduleName ( ) const [inline], [virtual]
```

Returns The name of the given module type as name.

Reimplemented from smtrat::Module.

15.323.4.73 objective()

```
carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]
```

15.323.4.74 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
    const FormulaT & _origin ) const [protected], [inherited]
```

15.323.4.75 passedFormulaBegin()

```
ModuleInput::iterator smtrat::Module::passedFormulaBegin ( )
[inline], [protected], [inherited]
```

Returns An iterator to the end of the passed formula. TODO: disable this method

15.323.4.76 passedFormulaEnd()

```
ModuleInput::iterator smtrat::Module::passedFormulaEnd ( )
[inline], [protected], [inherited]
```

Returns An iterator to the end of the passed formula.

15.323.4.77 pPassedFormula()

```
const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline],
[inherited]
```

Returns A pointer to the formula passed to the backends of this module.

15.323.4.78 pReceivedFormula()

```
const ModuleInput* smtrat::Module::pReceivedFormula ( ) const
[inline], [inherited]
```

Returns A pointer to the formula passed to this module.

15.323.4.79 print()

```
void smtrat::Module::print (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.323.4.80 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

_out	The stream to print the assignment on.

15.323.4.81 printInfeasibleSubsets()

`void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.323.4.82 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

_out	The stream to print the assignment on.

15.323.4.83 printPassedFormula()

`void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.323.4.84 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.323.4.85 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.

15.323.4.86 receivedFormulaChecked()

```
void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]
```

Notifies that the received formulas has been checked.

15.323.4.87 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩InfeasibleSubset (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

15.323.4.88 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const [inline], [inherited]
```

15.323.4.89 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

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

15.323.4.90 removeCore()

```
template<class 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

_subformula	The position of the subformula to remove.

Reimplemented from smtrat::Module.

15.323.4.91 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin
(
            ModuleInput::iterator _formula,
            const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.323.4.92 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

15.323.4.93 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline],
[inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.323.4.94 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.323.4.95 runBackends() [1/2]

`virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]`

Reimplemented in smtrat::PModule.

15.323.4.96 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

_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 in smtrat::PModule.

15.323.4.97 setId()

```
void smtrat::Module::setId (
    std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.323.4.98 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.

15.323.4.99 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.

15.323.4.100 splitUnequalConstraint()

`void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.323.4.101 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.323.4.102 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 >.

15.323.4.103 updateLemmas()

`void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.323.4.104 updateModel()

`template<class 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.

15.323.4.105 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).

15.323.5 Field Documentation

15.323.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.323.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.323.5.3 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.

15.323.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.323.5.5 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

15.323.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.323.5.7 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.

15.323.5.8 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.

15.323.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.323.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.323.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.323.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.323.5.13 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.

15.323.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.323.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.323.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.323.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.323.5.18 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.

15.323.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]`

Reusable splitting variables.

15.323.5.20 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.323.5.21 mSmallerMusesCheckCounter

`unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.323.5.22 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.

15.323.5.23 mTheoryPropagations

`std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]`

15.323.5.24 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).

15.323.5.25 mVariableCounters

`std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.324 smtrat::parser::NumeralParser Struct Reference

Parses numerals: `(0 | [1-9][0-9]*)`

```
#include <Lexicon.h>
```

15.324.1 Detailed Description Parses numerals: `(0 | [1-9][0-9]*)`

15.325 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 inteeger.
- 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 isPositive () const

  Checks whether this Numeric corresponds to a positive rational number.
- bool isNegative () const
- bool isZero () const
- bool isInteger () const

15.325.1 Constructor & Destructor Documentation

15.325.1.1 Numeric() [1/8] `smtrat::lra::Numeric::Numeric ( )`

Default constructor.

15.325.1.2 Numeric() [2/8] `smtrat::lra::Numeric::Numeric ( const Rational & _value )`

Constructing from a Rational.

Parameters

The	Rational.

15.325.1.3 Numeric() [3/8] `smtrat::lra::Numeric::Numeric ( int _value )`

Constructing from an integer.

Parameters

_value	The integer.

15.325.1.4 Numeric() [4/8] `smtrat::lra::Numeric::Numeric ( unsigned int _value )`

Constructing from an unsigned integer.

Parameters

_value	The unsigned integer

15.325.1.5 Numeric() [5/8] smtrat::lra::Numeric::Numeric (long _value)

Constructing from a long integer.

Parameters

_value	The unsigned long integer.

15.325.1.6 Numeric() [6/8] smtrat::lra::Numeric::Numeric (unsigned long _value)

Constructing from an unsigned long integer.

Parameters

_value	The unsigned long integer.

15.325.1.7 Numeric() [7/8] smtrat::lra::Numeric::Numeric (const char * _value)

Constructing from a char array.

Parameters

_value	The char array.

15.325.1.8 Numeric() [8/8] smtrat::lra::Numeric::Numeric (const Numeric & _value)

Copy constructor.

Parameters

_value	The Numeric to copy.

15.325.1.9 ~Numeric() smtrat::lra::Numeric::~Numeric ( )

15.325.2 Member Function Documentation

15.325.2.1 content() const Rational& smtrat::lra::Numeric::content ( ) const [inline]

15.325.2.2 denom() `Numeric smtrat::lra::Numeric::denom ( ) const`

Returns The denominator of this Numeric.

15.325.2.3 floor() `Numeric smtrat::lra::Numeric::floor ( ) const`

Returns The next smaller integer to this Numeric.

15.325.2.4 isInteger() `bool smtrat::lra::Numeric::isInteger ( ) const`

Returns true, if this Numeric is integer; false, otherwise.

15.325.2.5 isNegative() `bool smtrat::lra::Numeric::isNegative ( ) const`

Returns true, if this Numeric corresponds to a positive rational number; false, otherwise.

15.325.2.6 isPositive() `bool smtrat::lra::Numeric::isPositive ( ) const`

Checks whether this Numeric corresponds to a positive rational number.

Returns True, if this Numeric corresponds to a positive rational number; False, otherwise.

15.325.2.7 isZero() `bool smtrat::lra::Numeric::isZero ( ) const`

Returns true, if this Numeric corresponds to zero; false, otherwise.

15.325.2.8 numer() `Numeric smtrat::lra::Numeric::numer ( ) const`

Returns The enumerator of this Numeric.

15.325.2.9 operator"!=() `bool smtrat::lra::Numeric::operator!= ( const Numeric & _value ) const`

Compares whether this Numeric and the given one are not equal.

Parameters

_value	The Numeric to compare with.

Returns True, if the two Numerics are not equal; False, otherwise.

15.325.2.10 operator<() `bool smtrat::lra::Numeric::operator< ( const Numeric & _value ) const`

Compares whether this Numeric is less than the given one.

Parameters

_value	The Numeric to compare with.

Returns True, if this Numeric is less than the given one; False, otherwise.

15.325.2.11 operator<=() `bool smtrat::lra::Numeric::operator<= ( const Numeric & _value ) const`

Compares whether this Numeric is less or equal than the given one.

Parameters

_value	The Numeric to compare with.

Returns True, if this Numeric is less or equal than the given one; False, otherwise.

15.325.2.12 operator=() [1/6] `Numeric & smtrat::lra::Numeric::operator= ( const char * _value )`

Cast from a char array.

Parameters

_value	The char array.

Returns The corresponding Numeric.

15.325.2.13 operator=() [2/6]

```
Numeric & smtrat::lra::Numeric::operator= (
    const Numeric & _value )
```

Cast from a char array.

Parameters

_value	The char array.

Returns The corresponding Numeric.

15.325.2.14 operator=() [3/6]

```
Numeric & smtrat::lra::Numeric::operator= (
    int _value )
```

Cast from an integer.

Parameters

_value	The integer.

Returns The corresponding Numeric.

15.325.2.15 operator=() [4/6]

```
Numeric & smtrat::lra::Numeric::operator= (
    long _value )
```

Cast from a long integer.

Parameters

_value	The long integer.

Returns The corresponding Numeric.

15.325.2.16 operator=() [5/6]

```
Numeric & smtrat::lra::Numeric::operator= (
    unsigned int _value )
```

Cast from an unsigned inteeger.

Parameters

_value	The unsigned integer.

Returns The corresponding Numeric.

15.325.2.17 operator=() [6/6] Numeric & smtrat::lra::Numeric::operator= (unsigned long _value)

Cast from an unsigned long integer.

Parameters

_value	The unsigned long integer.

Returns The corresponding Numeric.

15.325.2.18 operator==() bool smtrat::lra::Numeric::operator== (const Numeric & _value) const

Compares whether this Numeric and the given one are equal.

Parameters

_value	The Numeric to compare with.

Returns True, if the two Numerics are equal; False, otherwise.

15.325.2.19 operator>() bool smtrat::lra::Numeric::operator> (const Numeric & _value) const

Compares whether this Numeric is greater than the given one.

Parameters

_value	The Numeric to compare with.

Returns True, if this Numeric is greater than the given one; False, otherwise.

15.325.2.20 operator>=() bool smtrat::lra::Numeric::operator>= (const Numeric & _value) const

Compares whether this Numeric is greater or equal than the given one.

Parameters

_value	The Numeric to compare with.

Returns True, if this Numeric is greater or equal than the given one; False, otherwise.

15.325.2.21 rContent() `Rational& smtrat::lra::Numeric::rContent ( ) [inline]`

15.326 smtrat::execution::Objective Struct Reference

`#include <ExecutionState.h>`

Data Fields

- Poly function
- bool minimize

15.326.1 Field Documentation

15.326.1.1 function `Poly smtrat::execution::Objective::function`

15.326.1.2 minimize `bool smtrat::execution::Objective::minimize`

15.327 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)

15.327.1 Constructor & Destructor Documentation

15.327.1.1 OccLists() `template<class Idx, class Vec, class Deleted>`

```
Minisat::OccLists< Idx, Vec, Deleted >::OccLists (
            const Deleted & d ) [inline]
```

15.327.2 Member Function Documentation

15.327.2.1 clean() `template<class Idx, class Vec , class Deleted >`

```
void Minisat::OccLists< Idx, Vec, Deleted >::clean (
            const Idx & idx )
```

15.327.2.2 cleanAll()

```
template<class Idx , class Vec , class Deleted >
void Minisat::OccLists< Idx, Vec, Deleted >::cleanAll ( )
```

15.327.2.3 clear()

```
template<class Idx, class Vec, class Deleted>
void Minisat::OccLists< Idx, Vec, Deleted >::clear (
            bool free = true )  [inline]
```

15.327.2.4 init()

```
template<class Idx, class Vec, class Deleted>
void Minisat::OccLists< Idx, Vec, Deleted >::init (
            const Idx & idx )  [inline]
```

15.327.2.5 lookup()

```
template<class Idx, class Vec, class Deleted>
Vec& Minisat::OccLists< Idx, Vec, Deleted >::lookup (
            const Idx & idx )  [inline]
```

15.327.2.6 operator[]() [1/2]

```
template<class Idx, class Vec, class Deleted>
Vec& Minisat::OccLists< Idx, Vec, Deleted >::operator[] (
            const Idx & idx )  [inline]
```

15.327.2.7 operator[]() [2/2]

```
template<class Idx, class Vec, class Deleted>
const Vec& Minisat::OccLists< Idx, Vec, Deleted >::operator[] (
            const Idx & idx ) const  [inline]
```

15.327.2.8 smudge()

```
template<class Idx, class Vec, class Deleted>
void Minisat::OccLists< Idx, Vec, Deleted >::smudge (
            const Idx & idx )  [inline]
```

15.328 smtrat::mcsat::onecellcad::OneCellCAD Class Reference

```
#include <utils.h>
```

Public Member Functions

- OneCellCAD (const std::vector< carl::Variable > &variableOrder, const carl::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 carl::RealAlgebraicPoint< Rational > & point

15.328.1 Constructor & Destructor Documentation

15.328.1.1 OneCellCAD()

```
smtrat::mcsat::onecellcad::OneCellCAD::OneCellCAD (
    const std::vector< carl::Variable > & variableOrder,
    const carl::RealAlgebraicPoint< Rational > & point ) [inline]
```

15.328.2 Member Function Documentation

15.328.2.1 coeffNonNull()

```
std::optional<Poly> smtrat::mcsat::onecellcad::OneCellCAD::coeff↩
NonNull (
    const TagPoly & boundCandidate ) [inline]
```

15.328.2.2 isMainPointInsideCell()

```
bool smtrat::mcsat::onecellcad::OneCellCAD::isMainPoint↩
InsideCell (
    const CADCell & cell ) [inline]
```

15.328.2.3 isolateLastVariableRoots()

```
std::vector<RAN> smtrat::mcsat::onecellcad::OneCellCAD↩
::isolateLastVariableRoots (
    const std::size_t polyLevel,
    const Poly & poly ) [inline]
```

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!).

15.328.2.4 isPointRootOfPoly() [1/2]

```
bool smtrat::mcsat::onecellcad::OneCellCAD::isPointRootOf↩
Poly (
    const std::size_t polyLevel,
    const Poly & poly ) [inline]
```

15.328.2.5 isPointRootOfPoly() [2/2]

```
bool smtrat::mcsat::onecellcad::OneCellCAD::isPointRootOf↩
Poly (
    const TagPoly & poly ) [inline]
```

15.328.2.6 prefixPointToStdMap()

```
std::map<carl::Variable, RAN> smtrat::mcsat::onecellcad::↩
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'.

15.328.2.7 vanishesEarly()

```
bool smtrat::mcsat::onecellcad::OneCellCAD::vanishesEarly (
    const std::size_t polyLevel,
    const Poly & poly ) [inline]
```

Check if an n-variate 'poly' p(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'.

15.328.3 Field Documentation

15.328.3.1 point

```
const carl::RealAlgebraicPoint<Rational>& smtrat::mcsat::onecellcad::OneCellCAD::point
```

15.328.3.2 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]

15.329 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.

15.329.1 Detailed Description

Operation settings.

15.329.2 Field Documentation

15.329.2.1 backend

```
std::string benchmax::settings::OperationSettings::backend
```

Name of the backend to be used.

15.329.2.2 convert_ods_filename

```
std::string benchmax::settings::OperationSettings::convert_ods_filename
```

Name of the xml file to convert to an ods file.

15.329.2.3 convert_ods_filter `std::string benchmax::settings::OperationSettings::convert_ods_filter`

Name of the xslt filter used for ods import.

15.329.2.4 mode `std::string benchmax::settings::OperationSettings::mode`

Name of the operation mode.

15.329.2.5 use_temp `bool benchmax::settings::OperationSettings::use_temp`

Use temporary directory.

15.330 smtrat::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)

15.330.1 Constructor & Destructor Documentation

15.330.1.1 Optimization()

```
template<typename Solver>
smtrat::Optimization< Solver >::Optimization (
            Solver & s )  [inline]
```

15.330.2 Member Function Documentation

15.330.2.1 active()

```
template<typename Solver>
bool smtrat::Optimization< Solver >::active ( ) const  [inline]
```

15.330.2.2 add_objective()

```
template<typename Solver>
void smtrat::Optimization< Solver >::add_objective (
            const Poly & objective,
            bool minimize = true )  [inline]
```

15.330.2.3 compute()

```
template<typename Solver>
std::tuple<Answer,Model,ObjectiveValues> smtrat::Optimization< Solver >::compute (
            bool full = true )  [inline]
```

15.330.2.4 remove_objective()

```
template<typename Solver>
void smtrat::Optimization< Solver >::remove_objective (
            const Poly & objective )  [inline]
```

15.330.2.5 reset() `template<typename Solver>`
`void smtrat::Optimization< Solver >::reset ( ) [inline]`

15.331 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)

15.331.1 Constructor & Destructor Documentation

15.331.1.1 Option()
```
Minisat::Option::Option (
    const char * name_,
    const char * desc_,
    const char * cate_,
    const char * type_ ) [inline], [protected]
```

15.331.1.2 ~Option()
`virtual Minisat::Option::~Option ( ) [inline], [virtual]`

15.331.2 Member Function Documentation

15.331.2.1 getHelpPrefixString()
`static const char*& Minisat::Option::getHelpPrefixString ( ) [inline], [static], [protected]`

15.331.2.2 getOptionList()
`static vec<Option*>& Minisat::Option::getOptionList ( ) [inline], [static], [protected]`

15.331.2.3 getUsageString()

`static const char*& Minisat::Option::getUsageString ( ) [inline], [static], [protected]`

15.331.2.4 help()

```
virtual void Minisat::Option::help (
    bool verbose = false ) [pure virtual]
```

Implemented in Minisat::BoolOption, Minisat::StringOption, Minisat::IntOption, and Minisat::DoubleOption.

15.331.3 Friends And Related Function Documentation

15.331.3.1 printUsageAndExit

```
void printUsageAndExit (
    int argc,
    char ** argv,
    bool verbose ) [friend]
```

15.331.3.2 setHelpPrefixStr

```
void setHelpPrefixStr (
    const char * str ) [friend]
```

15.331.3.3 setUsageHelp

```
void setUsageHelp (
    const char * str ) [friend]
```

15.331.4 Field Documentation

15.331.4.1 category

`const char* Minisat::Option::category [protected]`

15.331.4.2 description

`const char* Minisat::Option::description [protected]`

15.331.4.3 name

`const char* Minisat::Option::name [protected]`

15.331.4.4 type_name

`const char* Minisat::Option::type_name [protected]`

15.332 Minisat::Option::OptionLt Struct Reference

`#include <Options.h>`

Public Member Functions

- bool operator() (const Option *x, const Option *y)

15.332.1 Member Function Documentation

15.332.1.1 operator()()

```
bool Minisat::Option::OptionLt::operator() (
    const Option * x,
    const Option * y ) [inline]
```

15.333 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

15.333.1 Constructor & Destructor Documentation

15.333.1.1 OrderPair()

```
template<typename T1, typename T2, bool reversed = false>
smtrat::datastructures::OrderPair< T1, T2, reversed >::OrderPair (
            T1 pFirst = T1(),
            T2 pSecond = T2() )  [inline]
```

15.333.2 Member Function Documentation

15.333.2.1 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]
```

15.334 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

15.334.1 Constructor & Destructor Documentation

15.334.1.1 OrderPair()

```
template<typename T1 , typename T2 >
smtrat::datastructures::OrderPair< T1, T2, true >::OrderPair (
            T1 pFirst = T1(),
            T2 pSecond = T2() )  [inline]
```

15.334.2 Member Function Documentation

15.334.2.1 operator<()

```
template<typename T1 , typename T2 >
bool smtrat::datastructures::OrderPair< T1, T2, true >::operator< (
            const OrderPair< T1, T2, true > & rhs ) const  [inline]
```

15.335 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)

15.335.1 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.

15.335.2 Constructor & Destructor Documentation

15.335.2.1 Origin() [1/5]

```
smtrat::cad::Origin::Origin ( ) [inline]
```

15.335.2.2 Origin() [2/5]

```
smtrat::cad::Origin::Origin (
    const Origin & po ) [inline]
```

15.335.2.3 Origin() [3/5]

```
smtrat::cad::Origin::Origin (
    Origin && po ) [inline]
```

15.335.2.4 Origin() [4/5]

```
smtrat::cad::Origin::Origin (
    std::size_t level,
    std::size_t id ) [inline], [explicit]
```

15.335.2.5 Origin() [5/5]

```
smtrat::cad::Origin::Origin (
    const BaseType & bt ) [inline], [explicit]
```

15.335.3 Member Function Documentation

15.335.3.1 activate()

```
void smtrat::cad::Origin::activate (
    std::size_t level,
    const carl::Bitset & rhs )  [inline]
```

15.335.3.2 activateEC()

```
void smtrat::cad::Origin::activateEC (
    std::size_t level,
    const carl::Bitset & rhs )  [inline]
```

15.335.3.3 begin()

```
auto smtrat::cad::Origin::begin ( ) const  [inline]
```

15.335.3.4 deactivate()

```
void smtrat::cad::Origin::deactivate (
    std::size_t level,
    const carl::Bitset & rhs )  [inline]
```

15.335.3.5 deactivateEC()

```
void smtrat::cad::Origin::deactivateEC (
    std::size_t level,
    const carl::Bitset & rhs )  [inline]
```

15.335.3.6 empty()

```
bool smtrat::cad::Origin::empty ( ) const  [inline]
```

15.335.3.7 end()

```
auto smtrat::cad::Origin::end ( ) const  [inline]
```

15.335.3.8 erase()

```
Origin& smtrat::cad::Origin::erase (
    std::size_t level,
    const carl::Bitset & rhs )  [inline]
```

15.335.3.9 isActive()

```
bool smtrat::cad::Origin::isActive ( ) const  [inline]
```

15.335.3.10 operator+=()

```
Origin& smtrat::cad::Origin::operator+= (
    const BaseType & rhs )  [inline]
```

Adds the pair to the origins.

15.335.3.11 operator-=()

```
Origin& smtrat::cad::Origin::operator-= (
    const BaseType & rhs )  [inline]
```

Removes the pair from the origins.

15.335.3.12 operator=() [1/2]

```
Origin& smtrat::cad::Origin::operator= (
    const Origin & rhs ) [inline]
```

15.335.3.13 operator=() [2/2]

```
Origin& smtrat::cad::Origin::operator= (
    Origin && rhs ) [inline]
```

15.335.3.14 operator==()

```
bool smtrat::cad::Origin::operator== (
    const Origin & rhs ) const [inline]
```

15.335.3.15 operator"|()

```
Origin smtrat::cad::Origin::operator| (
    const Origin & rhs ) const [inline]
```

15.335.4 Friends And Related Function Documentation

15.335.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const Origin & po ) [friend]
```

15.336 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

15.336.1 Member Function Documentation

15.336.1.1 add()

```
void smtrat::cad::preprocessor::Origins::add (
    const FormulaT & f,
    const std::vector< FormulaT > & origin )
```

15.336.1.2 cleanOrigins()

```
void smtrat::cad::preprocessor::Origins::cleanOrigins (
    const FormulaT & f )
```

15.336.1.3 get()

```
const std::vector< FormulaT > & smtrat::cad::preprocessor::Origins::get (
    const FormulaT & f ) const
```

15.336.1.4 remove()

```
void smtrat::cad::preprocessor::Origins::remove (
    const FormulaT & f )
```

15.336.1.5 resolve()

```
bool smtrat::cad::preprocessor::Origins::resolve (
    std::set< FormulaT > & conflict ) const
```

15.336.2 Field Documentation

15.336.2.1 mOrigins

```
std::map<FormulaT, std::vector<std::vector<FormulaT> > > smtrat::cad↩
::preprocessor::Origins::mOrigins
```

15.337 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)

15.337.1 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).

15.337.2 Constructor & Destructor Documentation

15.337.2.1 OrPathShortener()

```
smtrat::OrPathShortener::OrPathShortener (
    ModuleWrapper< EQModule< EQSettingsForPreprocessing >> & facts ) [inline]
```

15.337.3 Member Function Documentation

15.337.3.1 enter_default()

```
void smtrat::OrPathShortener::enter_default (
    const FormulaT & formula ) [inline]
```

15.337.3.2 rewrite_default()

```
FormulaT smtrat::OrPathShortener::rewrite_default (
    const FormulaT & formula,
    const FormulaT & default_result ) [inline]
```

15.337.3.3 rewrite_or()

```
FormulaT smtrat::OrPathShortener::rewrite_or (
    const FormulaT & formula,
    FormulaSetT && subformulas ) [inline]
```

15.337.3.4 rewrite_xor()

```
FormulaT smtrat::OrPathShortener::rewrite_xor (
    const FormulaT & formula,
    FormulasMultiT && subformulas ) [inline]
```

15.338 Minisat::OutOfMemoryException Class Reference

```
#include <XAlloc.h>
```

15.339 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)

15.339.1 Constructor & Destructor Documentation

15.339.1.1 OutputWrapper() [1/2]

```
smtrat::parser::OutputWrapper::OutputWrapper (
    std::ostream & o,
    const std::string & prefix,
    const std::string & suffix ) [inline]
```

15.339.1.2 OutputWrapper() [2/2]

```
smtrat::parser::OutputWrapper::OutputWrapper (
    const OutputWrapper && o ) [inline]
```

15.339.1.3 ~OutputWrapper()

```
smtrat::parser::OutputWrapper::~OutputWrapper ( ) [inline]
```

15.339.2 Member Function Documentation

15.339.2.1 operator<<()

```
template<typename T >
OutputWrapper& smtrat::parser::OutputWrapper::operator<< (
    const T & t ) [inline]
```

15.340 Minisat::Map< K, D, H, E >::Pair Struct Reference

```
#include <Map.h>
```

Data Fields

- K key
- D data

15.340.1 Field Documentation

15.340.1.1 data

```
template<class K, class D, class H = Hash<K>, class E = Equal<K>>
D Minisat::Map< K, D, H, E >::Pair::data
```

15.340.1.2 key

```
template<class K, class D, class H = Hash<K>, class E = Equal<K>>
K Minisat::Map< K, D, H, E >::Pair::key
```

15.341 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

15.341.1 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.

15.341.2 Member Function Documentation

15.341.2.1 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]
```

15.342 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

15.342.1 Member Function Documentation

15.342.1.1 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]
```

15.343 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.

15.343.1 Detailed Description This class parses a whole smtlib file into a hierarchy of Node objects.

15.343.2 Constructor & Destructor Documentation

15.343.2.1 Parser() `delta::Parser::Parser ( ) [inline]`

Constructs the parsing rules.

15.343.3 Member Function Documentation

15.343.3.1 parse()

```
static bool delta::Parser::parse (
    const std::string & filename,
    Node & node ) [inline], [static]
```

Parses a file into a node.

Parameters

filename	Filename of the input file.

Returns Node object produced by the parser.

15.343.3.2 parseFile()

```
bool delta::Parser::parseFile (
    const std::string & filename,
    Node & node ) [inline]
```

Parses a file into a node.

Parameters

filename	Filename of the input file.

Returns Node object produced by the parser.

15.344 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

15.344.1 Field Documentation

15.344.1.1 disable_theory `bool smtrat::parser::ParserSettings::disable_theory`

15.344.1.2 disable_uf_flattening `bool smtrat::parser::ParserSettings::disable_uf_flattening`

15.344.1.3 input_file `std::string smtrat::parser::ParserSettings::input_file`

15.344.1.4 read_dimacs `bool smtrat::parser::ParserSettings::read_dimacs`

15.344.1.5 read_opb `bool smtrat::parser::ParserSettings::read_opb`

15.345 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

15.345.1 Constructor & Destructor Documentation

15.345.1.1 ParserState()

```
smtrat::parser::ParserState::ParserState (
    InstructionHandler & ih ) [inline]
```

15.345.1.2 ~ParserState()

```
smtrat::parser::ParserState::~ParserState ( ) [inline]
```

15.345.2 Member Function Documentation

15.345.2.1 errorMessage()

```
void smtrat::parser::ParserState::errorMessage (
    const std::string & msg ) [inline]
```

15.345.2.2 isSymbolFree()

```
bool smtrat::parser::ParserState::isSymbolFree (
    const std::string & name,
    bool output = true ) [inline]
```

15.345.2.3 popExpressionScope()

```
void smtrat::parser::ParserState::popExpressionScope ( )
[inline]
```

15.345.2.4 popScriptScope()

```
void smtrat::parser::ParserState::popScriptScope ( ) [inline]
```

15.345.2.5 pushExpressionScope()

```
void smtrat::parser::ParserState::pushExpressionScope ( )
[inline]
```

15.345.2.6 pushScriptScope()

```
void smtrat::parser::ParserState::pushScriptScope ( ) [inline]
```

15.345.2.7 registerFunction() [1/3]

```
void smtrat::parser::ParserState::registerFunction (
    const std::string & name,
    const FunctionInstantiator * fi ) [inline]
```

15.345.2.8 registerFunction() [2/3]

```
void smtrat::parser::ParserState::registerFunction (
    const std::string & name,
    const IndexedFunctionInstantiator * fi ) [inline]
```

15.345.2.9 registerFunction() [3/3]

```
void smtrat::parser::ParserState::registerFunction (
    const std::string & name,
    const UserFunctionInstantiator * fi ) [inline]
```

15.345.2.10 reset()

```
void smtrat::parser::ParserState::reset ( ) [inline]
```

15.345.2.11 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]
```

15.345.2.12 resolveSymbol() [2/3]

```
bool smtrat::parser::ParserState::resolveSymbol (
            const std::string & name,
            types::TermType & r ) const [inline]
```

15.345.2.13 resolveSymbol() [3/3]

```
bool smtrat::parser::ParserState::resolveSymbol (
            const std::string & name,
            types::VariableType & r ) const [inline]
```

15.345.2.14 script_scope_size()

```
std::size_t smtrat::parser::ParserState::script_scope_size ( )
const [inline]
```

15.345.3 Field Documentation

15.345.3.1 artificialVariables

```
std::vector<smtrat::ModelVariable> smtrat::parser::ParserState↩
::artificialVariables
```

15.345.3.2 auxiliary_variables

```
std::set<types::VariableType> smtrat::parser::ParserState::auxiliary↩
_variables
```

15.345.3.3 bindings

```
std::map<std::string, types::TermType> smtrat::parser::ParserState↩
::bindings
```

15.345.3.4 constants

```
std::map<std::string, types::TermType> smtrat::parser::ParserState↩
::constants
```

15.345.3.5 declared_functions

```
std::map<std::string, carl::UninterpretedFunction> smtrat↩
::parser::ParserState::declared_functions
```

15.345.3.6 defined_functions

```
std::map<std::string, const FunctionInstantiator*> smtrat::parser↩
::ParserState::defined_functions
```

15.345.3.7 defined_indexed_functions

`std::map<std::string, const IndexedFunctionInstantiator*> smtrat::parser::ParserState::defined_indexed_functions`

15.345.3.8 defined_user_functions

`std::map<std::string, const UserFunctionInstantiator*> smtrat↩::parser::ParserState::defined_user_functions`

15.345.3.9 expressionScopes

`std::stack<ExpressionScope> smtrat::parser::ParserState::expression↩Scopes`

15.345.3.10 global_formulas

`FormulasT smtrat::parser::ParserState::global_formulas`

15.345.3.11 handler

`InstructionHandler& smtrat::parser::ParserState::handler`

15.345.3.12 logic

`carl::Logic smtrat::parser::ParserState::logic`

15.345.3.13 scriptScopes

`std::stack<ScriptScope> smtrat::parser::ParserState::scriptScopes`

15.345.3.14 variables

`std::map<std::string, types::VariableType> smtrat::parser::ParserState↩::variables`

15.346 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 LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::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 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 _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 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 &_branching↩Value) 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 _preferLeft↩Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩::vector< FormulaT >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩CaseWeak=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.

15.346.1 Member Typedef Documentation

15.346.1.1 MatrixT

```
template<typename Settings >
using smtrat::PBGaussModule< Settings >::MatrixT = Eigen::Matrix<Rational, Eigen::Dynamic,
Eigen::Dynamic>
```

15.346.1.2 SettingsType

```
template<typename Settings >
typedef Settings smtrat::PBGaussModule< Settings >::SettingsType
```

15.346.1.3 VectorT

```
template<typename Settings >
using smtrat::PBGaussModule< Settings >::VectorT = Eigen::Matrix<Rational, Eigen::Dynamic, 1>
```

15.346.2 Member Enumeration Documentation

15.346.2.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

15.346.3 Constructor & Destructor Documentation

15.346.3.1 PBGaussModule()

```
template<class Settings >
smtrat::PBGaussModule< Settings >::PBGaussModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

15.346.3.2 ~PBGaussModule()

```
template<class Settings >
smtrat::PBGaussModule< Settings >::~PBGaussModule ( )
```

15.346.4 Member Function Documentation

15.346.4.1 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.

15.346.4.2 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 >.

15.346.4.3 addCore()

```
template<class 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

_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.

15.346.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.346.4.5 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.

15.346.4.6 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.

15.346.4.7 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.

15.346.4.8 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.

15.346.4.9 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.

15.346.4.10 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

_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.

15.346.4.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns All satisfying assignments, if existent.

15.346.4.12 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.

15.346.4.13 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.

15.346.4.14 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.

15.346.4.15 branchAt() [1/6]

```
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]
```

15.346.4.16 branchAt() [2/6]

```
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]
```

15.346.4.17 branchAt() [3/6]

```
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]
```

15.346.4.18 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.346.4.19 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            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]
```

15.346.4.20 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

15.346.4.21 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in smtrat::PModule.

15.346.4.22 checkCore()

```
template<class Settings >
Answer smtrat::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 smtrat::Module.

15.346.4.23 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
    std::vector< FormulaSetT >::const_iterator _infsubset,
    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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.346.4.24 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.

15.346.4.25 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.346.4.26 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.346.4.27 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.346.4.28 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.346.4.29 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.346.4.30 collectOrigins() [1/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

15.346.4.31 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

_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.

15.346.4.32 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.346.4.33 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.

15.346.4.34 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.

15.346.4.35 constraintsToInform()

```
const carl::FastSet<FormulaT>& smtrat::Module::constraints↩
ToInform ( ) const [inline], [inherited]
```

Returns The constraints which the backends must be informed about.

15.346.4.36 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::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.346.4.37 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)

15.346.4.38 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

_constraint	The constraint to remove from internal data structures.

15.346.4.39 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.346.4.40 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.346.4.41 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.346.4.42 excludeNotReceivedVariablesFromModel()

`void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

15.346.4.43 findBestOrigin()

`std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin ( const std::vector< FormulaT > & _origins ) const [protected], [inherited]`

Parameters

_origins	

Returns

15.346.4.44 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.

15.346.4.45 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.

15.346.4.46 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

15.346.4.47 gaussAlgorithm()

`template<class Settings > FormulaT smtrat::PBGaussModule< Settings >::gaussAlgorithm ( )`

15.346.4.48 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

15.346.4.49 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

15.346.4.50 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

15.346.4.51 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

15.346.4.52 getInfeasibleSubsets() [2/2] `std::vector< FormulaSetT > smtrat::Module::getInfeasible↩ Subsets ( 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.346.4.53 getInformationRelevantFormulas() `const std::set< FormulaT > & smtrat::Module::get↩ InformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.346.4.54 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.

15.346.4.55 getOrigins() [1/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

15.346.4.56 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.346.4.57 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.346.4.58 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 reveived formula, if the Boolean is true.

Reimplemented in smtrat::PModule.

15.346.4.59 hasLemmas()

```
bool smtrat::Module::hasLemmas ( ) [inline], [inherited]
```

Checks whether this module or any of its backends provides any lemmas.

15.346.4.60 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).

15.346.4.61 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns A unique ID to identify this module instance.

15.346.4.62 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.

15.346.4.63 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.346.4.64 informBackends() `void smtrat::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.

15.346.4.65 informCore() template<class Settings >

```
bool smtrat::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

_constraint	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.

15.346.4.66 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const  [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.346.4.67 init() template<class Settings >

```
void smtrat::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 smtrat::Module.

15.346.4.68 is_minimizing()

```
bool smtrat::Module::is_minimizing ( ) const  [inline], [inherited]
```

15.346.4.69 isLemmaLevel()

```
bool smtrat::Module::isLemmaLevel (
            LemmaLevel level )  [protected], [inherited]
```

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.346.4.70 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().

15.346.4.71 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.

15.346.4.72 lookForReductionRow()

```
template<typename Settings >
std::vector<Rational> smtrat::PBGaussModule< Settings >::lookForReductionRow (
            const MatrixT & uMatrix,
            const VectorT & ineqRow,
            long column )
```

15.346.4.73 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.346.4.74 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.346.4.75 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.

15.346.4.76 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.

15.346.4.77 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

15.346.4.78 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

15.346.4.79 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula. TODO: disable this method

15.346.4.80 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula.

15.346.4.81 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to the backends of this module.

15.346.4.82 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to this module.

15.346.4.83 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.346.4.84 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

_out	The stream to print the assignment on.

15.346.4.85 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.346.4.86 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

_out	The stream to print the assignment on.

15.346.4.87 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.346.4.88 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.346.4.89 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.

15.346.4.90 receivedFormulaChecked()

```
void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]
```

Notifies that the received formulas has been checked.

15.346.4.91 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩InfeasibleSubset (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

15.346.4.92 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const [inline], [inherited]
```

15.346.4.93 reconstructEqSystem()

```
template<class Settings >
FormulaT smtrat::PBGaussModule< Settings >::reconstructEqSystem (
    const MatrixT & m,
    const VectorT & b,
    const carl::Variables & vars,
    const std::vector< carl::Relation > & rels )
```

15.346.4.94 reduce()

```
template<class Settings >
FormulaT smtrat::PBGaussModule< Settings >::reduce (
            const MatrixT & u,
            const VectorT & b,
            const carl::Variables vars )
```

15.346.4.95 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.

15.346.4.96 removeCore()

```
template<class Settings >
void smtrat::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

_subformula	The position of the subformula to remove.

Reimplemented from smtrat::Module.

15.346.4.97 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin
(
            ModuleInput::iterator _formula,
            const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.346.4.98 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

15.346.4.99 rPassedFormula() `const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]`

Returns A reference to the formula passed to the backends of this module.

15.346.4.100 rReceivedFormula() `const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]`

Returns A reference to the formula passed to this module.

15.346.4.101 runBackends() [1/2] `virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]`

Reimplemented in smtrat::PModule.

15.346.4.102 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

_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 in smtrat::PModule.

15.346.4.103 setId()

```
void smtrat::Module::setId (
    std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ _↩ *id*	The id to set this module's id to.

15.346.4.104 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

_threadPriority	The priority to set this module's thread priority to.

15.346.4.105 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.

15.346.4.106 splitUnequalConstraint()

`void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.346.4.107 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.346.4.108 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 >.

15.346.4.109 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.346.4.110 updateModel() `template<class 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.

15.346.4.111 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).

15.346.5 Field Documentation

15.346.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.346.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.346.5.3 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.

15.346.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.346.5.5 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

15.346.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.346.5.7 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.

15.346.5.8 mFirstUncheckedReceivedSubformula `ModuleInput::const_iterator smtrat::Module::m↩ FirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

15.346.5.9 mFoundAnswer `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.346.5.10 mFullCheck `bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.346.5.11 mInfeasibleSubsets `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.346.5.12 mInformedConstraints `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.346.5.13 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.

15.346.5.14 mLemmas `std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.346.5.15 mModel `Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.346.5.16 mModelComputed `bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.346.5.17 mNumOfBranchVarsToStore `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.346.5.18 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.

15.346.5.19 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

15.346.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.346.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.346.5.22 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.

15.346.5.23 mTheoryPropagations

`std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]`

15.346.5.24 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).

15.346.5.25 mVariableCounters

`std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.347 smtrat::PBPPModule< Settings > Class Template Reference

`#include <PBPPModule.h>`

Public Types

- typedef Settings SettingsType
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::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 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 _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 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 &_branching↩ Value) 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 _preferLeft↩ Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩ ::vector< FormulaT >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩ AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩ CaseWeak=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.

15.347.1 Member Typedef Documentation

15.347.1.1 SettingsType `template<typename Settings >`

```
typedef Settings smtrat::PBPPModule< Settings >::SettingsType
```

15.347.2 Member Enumeration Documentation

15.347.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

15.347.3 Constructor & Destructor Documentation

15.347.3.1 PBPPModule() `template<class Settings >`

```
smtrat::PBPPModule< Settings >::PBPPModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

15.347.3.2 ~PBPPModule() `template<class Settings >`

```
smtrat::PBPPModule< Settings >::~PBPPModule ( )
```

15.347.4 Member Function Documentation

15.347.4.1 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.

15.347.4.2 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 >.

15.347.4.3 addCore()

```
template<class 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

_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.

15.347.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.347.4.5 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.

15.347.4.6 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.

15.347.4.7 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.

15.347.4.8 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.

15.347.4.9 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.

15.347.4.10 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

_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.

15.347.4.11 allModels() `const std::vector<Model>& smtrat::Module::allModels ( ) const [inline], [inherited]`

Returns All satisfying assignments, if existent.

15.347.4.12 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.

15.347.4.13 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.

15.347.4.14 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.

15.347.4.15 branchAt() [1/6]

```
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]
```

15.347.4.16 branchAt() [2/6]

```
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]
```

15.347.4.17 branchAt() [3/6]

```
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]
```

15.347.4.18 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.347.4.19 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
    const typename P::PolyType & _poly,
    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]
```

15.347.4.20 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false )  [inline], [protected], [inherited]
```

15.347.4.21 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in smtrat::PModule.

15.347.4.22 checkCore()

```
template<class Settings >
Answer smtrat::PBPPModule< 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.

15.347.4.23 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
            std::vector< FormulaSetT >::const_iterator _infsubset,
            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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.347.4.24 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.

15.347.4.25 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.347.4.26 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.347.4.27 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.347.4.28 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.347.4.29 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.347.4.30 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

15.347.4.31 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

_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.

15.347.4.32 collectTheoryPropagations()

`void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.347.4.33 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ToInform ( ) const [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.347.4.34 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

15.347.4.35 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)

15.347.4.36 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

_constraint	The constraint to remove from internal data structures.

15.347.4.37 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.347.4.38 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.347.4.39 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.347.4.40 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.347.4.41 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.347.4.42 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.

15.347.4.43 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.

15.347.4.44 freeSplittingVariable()

```
static void smtrat::Module::freeSplittingVariable (
    const FormulaT & _splittingVariable ) [inline], [static], [inherited]
```

15.347.4.45 generateTrivialInfeasibleSubset()

```
void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]
```

Stores the trivial infeasible subset being the set of received formulas.

15.347.4.46 getBackendsAllModels()

```
void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]
```

Stores all models of a backend in the list of all models of this module.

15.347.4.47 getBackendsModel()

```
void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]
```

15.347.4.48 getInfeasibleSubsets() [1/2]

```
void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]
```

Copies the infeasible subsets of the passed formula.

15.347.4.49 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.347.4.50 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.347.4.51 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.

15.347.4.52 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.347.4.53 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.347.4.54 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.347.4.55 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 reveived formula, if the Boolean is true.

Reimplemented in smtrat::PModule.

15.347.4.56 hasLemmas() bool smtrat::Module::hasLemmas ( ) [inline], [inherited]

Checks whether this module or any of its backends provides any lemmas.

15.347.4.57 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).

15.347.4.58 id() std::size_t smtrat::Module::id ( ) const [inline], [inherited]

Returns A unique ID to identify this module instance.

15.347.4.59 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.

15.347.4.60 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.347.4.61 informBackends()

```
void smtrat::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.

15.347.4.62 informCore()

```
template<class 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

_constraint	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.

15.347.4.63 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.347.4.64 init()

```
template<class Settings >
void smtrat::PBPPModule< 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.

15.347.4.65 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.347.4.66 isLemmaLevel() `bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.347.4.67 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().

15.347.4.68 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.

15.347.4.69 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.347.4.70 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.347.4.71 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

_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.

15.347.4.72 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.

15.347.4.73 objective()

```
carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]
```

15.347.4.74 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
    const FormulaT & _origin ) const [protected], [inherited]
```

15.347.4.75 passedFormulaBegin()

```
ModuleInput::iterator smtrat::Module::passedFormulaBegin ( )
[inline], [protected], [inherited]
```

Returns An iterator to the end of the passed formula. TODO: disable this method

15.347.4.76 passedFormulaEnd()

```
ModuleInput::iterator smtrat::Module::passedFormulaEnd ( )
[inline], [protected], [inherited]
```

Returns An iterator to the end of the passed formula.

15.347.4.77 pPassedFormula()

```
const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline],
[inherited]
```

Returns A pointer to the formula passed to the backends of this module.

15.347.4.78 pReceivedFormula()

```
const ModuleInput* smtrat::Module::pReceivedFormula ( ) const
[inline], [inherited]
```

Returns A pointer to the formula passed to this module.

15.347.4.79 print()

```
void smtrat::Module::print (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.347.4.80 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

_out	The stream to print the assignment on.

15.347.4.81 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.347.4.82 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

_out	The stream to print the assignment on.

15.347.4.83 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.347.4.84 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.347.4.85 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.

15.347.4.86 receivedFormulaChecked()

```
void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]
```

Notifies that the received formulas has been checked.

15.347.4.87 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩InfeasibleSubset (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

15.347.4.88 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const [inline], [inherited]
```

15.347.4.89 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

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

15.347.4.90 removeCore()

```
template<class Settings >
void smtrat::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

_subformula	The position of the subformula to remove.

Reimplemented from smtrat::Module.

15.347.4.91 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin
(
            ModuleInput::iterator _formula,
            const FormulaT & _origin )  [inline], [protected], [inherited]
```

15.347.4.92 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins )  [inline], [protected],
[inherited]
```

15.347.4.93 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const  [inline],
[inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.347.4.94 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.347.4.95 runBackends() [1/2]

`virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]`

Reimplemented in smtrat::PModule.

15.347.4.96 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

_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 in smtrat::PModule.

15.347.4.97 setId()

```
void smtrat::Module::setId (
    std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.347.4.98 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.

15.347.4.99 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.

15.347.4.100 splitUnequalConstraint()

`void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.347.4.101 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.347.4.102 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 >.

15.347.4.103 updateLemmas()

`void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.347.4.104 updateModel()

`template<class 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.

15.347.4.105 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).

15.347.5 Field Documentation

15.347.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.347.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.347.5.3 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.

15.347.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.347.5.5 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

15.347.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.347.5.7 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.

15.347.5.8 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.

15.347.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.347.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.347.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.347.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.347.5.13 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.

15.347.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.347.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.347.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.347.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.347.5.18 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.

15.347.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]`

Reusable splitting variables.

15.347.5.20 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.347.5.21 mSmallerMusesCheckCounter

`unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.347.5.22 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.

15.347.5.23 mTheoryPropagations

`std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]`

15.347.5.24 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).

15.347.5.25 mVariableCounters

`std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.348 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

15.348.1 Member Typedef Documentation

15.348.1.1 FindState

`using smtrat::PersistentUnionFind::FindState = std::pair<Parents, Representative>`

15.348.1.2 Parents

`using smtrat::PersistentUnionFind::Parents = immer::vector<Value>`

15.348.1.3 Ranks

`using smtrat::PersistentUnionFind::Ranks = immer::vector<size_t>`

15.348.1.4 Representative

`using smtrat::PersistentUnionFind::Representative = Value`

15.348.1.5 Value

`using smtrat::PersistentUnionFind::Value = size_t`

15.348.2 Member Function Documentation

15.348.2.1 dump()

`void smtrat::PersistentUnionFind::dump ( ) const [inline], [noexcept]`

15.348.2.2 find()

```
Representative smtrat::PersistentUnionFind::find (
    Value const & val ) [inline], [noexcept]
```

15.348.2.3 find_impl()

```
FindState smtrat::PersistentUnionFind::find_impl (
    Parents parents,
    Value const & i ) const [inline], [noexcept]
```

15.348.2.4 init_size()

```
void smtrat::PersistentUnionFind::init_size (
    size_t size ) [inline], [noexcept]
```

15.348.2.5 merge()

```
PersistentUnionFind smtrat::PersistentUnionFind::merge (
    Value const & a,
    Value const & b ) [inline], [noexcept]
```

15.348.2.6 resize()

```
void smtrat::PersistentUnionFind::resize (
    size_t size ) [inline], [noexcept]
```

15.348.2.7 update()

```
template<typename Ps , typename Rs >
void smtrat::PersistentUnionFind::update (
    Ps && parents,
    Rs && ranks ) [inline], [noexcept]
```

15.348.2.8 update_parents() [1/2]

```
void smtrat::PersistentUnionFind::update_parents (
    Parents && parents ) [inline], [noexcept]
```

15.348.2.9 update_parents() [2/2]

```
void smtrat::PersistentUnionFind::update_parents (
    Parents const & parents ) [inline], [noexcept]
```

15.348.2.10 update_ranks() [1/2]

```
void smtrat::PersistentUnionFind::update_ranks (
    Ranks && ranks ) [inline], [noexcept]
```

15.348.2.11 update_ranks() [2/2]

```
void smtrat::PersistentUnionFind::update_ranks (
    Ranks const & ranks ) [inline], [noexcept]
```

15.348.3 Field Documentation

15.348.3.1 _parents

```
Parents smtrat::PersistentUnionFind::_parents
```

15.348.3.2 _ranks

```
Ranks smtrat::PersistentUnionFind::_ranks
```

15.349 smtrat::PFEModule< Settings > Class Template Reference

```
#include <PFEModule.h>
```

Public Types

- typedef Settings SettingsType
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::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 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
- 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 _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.

- 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 &_branching↩Value) 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 _preferLeft↩Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩::vector< FormulaT >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩CaseWeak=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.

15.349.1 Member Typedef Documentation

15.349.1.1 SettingsType

```
template<typename Settings >
typedef Settings smtrat::PFEModule< Settings >::SettingsType
```

15.349.2 Member Enumeration Documentation

15.349.2.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

15.349.3 Constructor & Destructor Documentation

15.349.3.1 PFEModule()

```
template<class Settings >
smtrat::PFEModule< Settings >::PFEModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = NULL )
```

15.349.3.2 ~PFEModule()

```
template<class Settings >
smtrat::PFEModule< Settings >::~PFEModule ( )
```

15.349.4 Member Function Documentation

15.349.4.1 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

_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.

15.349.4.2 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 >.

15.349.4.3 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.

15.349.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
            const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.349.4.5 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.

15.349.4.6 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.

15.349.4.7 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.

15.349.4.8 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.

15.349.4.9 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.

15.349.4.10 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

_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.

15.349.4.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns All satisfying assignments, if existent.

15.349.4.12 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.

15.349.4.13 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.

15.349.4.14 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.

15.349.4.15 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.

15.349.4.16 branchAt() [1/6]

```
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]
```

15.349.4.17 branchAt() [2/6]

```
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]
```

15.349.4.18 branchAt() [3/6]

```
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]
```

15.349.4.19 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.349.4.20 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            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]
```

15.349.4.21 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

15.349.4.22 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

15.349.4.23 checkCore()

```
template<class 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.

15.349.4.24 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
    std::vector< FormulaSetT >::const_iterator _infsubset,
    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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.349.4.25 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.

15.349.4.26 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.349.4.27 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.349.4.28 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.349.4.29 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.349.4.30 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.349.4.31 collectOrigins() [1/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

15.349.4.32 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.

15.349.4.33 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.349.4.34 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraintsToInform ( ) const [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.349.4.35 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

15.349.4.36 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)

15.349.4.37 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

_constraint	The constraint to remove from internal data structures.

15.349.4.38 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.349.4.39 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.349.4.40 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.349.4.41 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceived↩
VariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.349.4.42 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBest↩
Origin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.349.4.43 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.

15.349.4.44 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.

15.349.4.45 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

15.349.4.46 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

15.349.4.47 getBackendsAllModels()

`void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

15.349.4.48 getBackendsModel()

`void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

15.349.4.49 getInfeasibleSubsets() [1/2]

`void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

15.349.4.50 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.349.4.51 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.349.4.52 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.

15.349.4.53 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.349.4.54 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.349.4.55 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.349.4.56 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 reveived formula, if the Boolean is true.

Reimplemented from smtrat::Module.

15.349.4.57 hasLemmas()

`bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

15.349.4.58 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).

15.349.4.59 id()

`std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns A unique ID to identify this module instance.

15.349.4.60 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.

15.349.4.61 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.349.4.62 informBackends()

`void smtrat::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.

15.349.4.63 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, and smtrat::ICPModule< ICPSettings1 >.

15.349.4.64 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.

15.349.4.65 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::FPPModule< Settings >.

15.349.4.66 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.349.4.67 isLemmaLevel() `bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.349.4.68 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().

15.349.4.69 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.

15.349.4.70 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.349.4.71 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.349.4.72 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.

15.349.4.73 moduleName() `virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]`

Returns The name of the given module type as name.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::EQPreprocessingModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::VSModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::GBModule< Settings >, smtrat::IntEqModule< Setti smtrat::CubeLIAModule< Settings >, smtrat::CurryModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IncWidthModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::BEModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::SplitSOSModule< Settings >.

15.349.4.74 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

15.349.4.75 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

15.349.4.76 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula. TODO: disable this method

15.349.4.77 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula.

15.349.4.78 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to the backends of this module.

15.349.4.79 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to this module.

15.349.4.80 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.349.4.81 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

_out	The stream to print the assignment on.

15.349.4.82 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.349.4.83 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

_out	The stream to print the assignment on.

15.349.4.84 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.349.4.85 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.349.4.86 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.

15.349.4.87 receivedFormulaChecked()

```
void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]
```

Notifies that the received formulas has been checked.

15.349.4.88 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩InfeasibleSubset (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

15.349.4.89 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const [inline], [inherited]
```

15.349.4.90 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

_subformula	The sub formula of the received formula to remove.

Reimplemented from smtrat::Module.

15.349.4.91 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

formula	The sub formula of the received formula to remove.

Reimplemented from smtrat::Module.

15.349.4.92 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin
(
            ModuleInput::iterator _formula,
            const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.349.4.93 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

15.349.4.94 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline],
[inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.349.4.95 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.349.4.96 runBackends() [1/2]

```
virtual Answer smtrat::PModule::runBackends ( ) [inline],
[virtual], [inherited]
```

Reimplemented from smtrat::Module.

15.349.4.97 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.

15.349.4.98 setId()

```
void smtrat::Module::setId (
    std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.349.4.99 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.

15.349.4.100 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.

15.349.4.101 splitUnequalConstraint()

```
void smtrat::Module::splitUnequalConstraint (
    const FormulaT & _unequalConstraint ) [protected], [inherited]
```

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.349.4.102 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.349.4.103 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 >.

15.349.4.104 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.349.4.105 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.

15.349.4.106 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).

15.349.5 Field Documentation

15.349.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.349.5.2 mAllModels

`std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.349.5.3 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.

15.349.5.4 mConstraintsToInform

`carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.349.5.5 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

15.349.5.6 mFirstPosInLastBranches

`std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.349.5.7 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.

15.349.5.8 mFirstUncheckedReceivedSubformula

`ModuleInput::const_iterator smtrat::Module::m↩FirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

15.349.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.349.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.349.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.349.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.349.5.13 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.

15.349.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.349.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.349.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.349.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.349.5.18 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.

15.349.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplitting↩Variables [static], [inherited]`

Reusable splitting variables.

15.349.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.349.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.349.5.22 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.

15.349.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

15.349.5.24 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).

15.349.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.350 smtrat::PModule Class Reference

`#include <PModule.h>`

Public Types

- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::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 smtrat::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 _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.

- 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
- 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 &_branching↩ Value) 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 _preferLeft↩ Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩ ::vector< FormulaT >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩ AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩ CaseWeak=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.

15.350.1 Member Enumeration Documentation

15.350.1.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

15.350.2 Constructor & Destructor Documentation

15.350.2.1 PModule()

```
smtrat::PModule::PModule (
    const ModuleInput * _formula,
    Conditionals & _foundAnswer,
    Manager * _manager = nullptr,
    std::string module_name = "PModule" )
```

15.350.3 Member Function Documentation

15.350.3.1 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

_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.

15.350.3.2 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 >.

15.350.3.3 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 smtrat::SATModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::VSModule< Settings >, smtrat::PFEModule< Settings >, smtrat::EQModule< Setting smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Sett smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, and smtrat::GBModule< Settings >.

15.350.3.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.350.3.5 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.

15.350.3.6 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.

15.350.3.7 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.

15.350.3.8 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.

15.350.3.9 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

_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.

15.350.3.10 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

_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.

15.350.3.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns All satisfying assignments, if existent.

15.350.3.12 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.

15.350.3.13 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.

15.350.3.14 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.

15.350.3.15 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.

15.350.3.16 branchAt() [1/6]

```
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]
```

15.350.3.17 branchAt() [2/6]

```
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]
```

15.350.3.18 branchAt() [3/6]

```
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]
```

15.350.3.19 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.

Parameters

_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.350.3.20 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            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]
```

15.350.3.21 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

15.350.3.22 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

15.350.3.23 checkCore()

```
Answer smtrat::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 smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::SATModule< Settings >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRAS smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::EQPreprocessingModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::VSModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::GBModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, smtrat::LVEModule< Settings >, smtrat::ESModule< Settings >, smtrat::GBPPModule< Settings >, smtrat::CNFerModule, smtrat::BEModule< Settings >, smtrat::MCBModule< Settings >, smtrat::PFEModule< Settings >, smtrat::SplitSOSModule< Setti smtrat::EMModule< Settings >, and smtrat::SymmetryModule< Settings >.

15.350.3.24 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
    std::vector< FormulaSetT >::const_iterator _infsubset,
    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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.350.3.25 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.

15.350.3.26 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.350.3.27 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.350.3.28 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.350.3.29 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.350.3.30 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.350.3.31 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

15.350.3.32 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.

15.350.3.33 collectTheoryPropagations()

`void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.350.3.34 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.350.3.35 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

15.350.3.36 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)

15.350.3.37 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

_constraint	The constraint to remove from internal data structures.

15.350.3.38 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.350.3.39 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.350.3.40 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.350.3.41 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceived↩
VariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.350.3.42 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBest↩
Origin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.350.3.43 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.

15.350.3.44 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.

15.350.3.45 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

15.350.3.46 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

15.350.3.47 getBackendsAllModels()

`void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

15.350.3.48 getBackendsModel()

`void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

15.350.3.49 getInfeasibleSubsets() [1/2]

`void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

15.350.3.50 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.350.3.51 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.350.3.52 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.

15.350.3.53 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.350.3.54 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.350.3.55 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.350.3.56 getReceivedFormulaSimplified()

`std::pair< bool, FormulaT > smtrat::PModule::getReceivedFormulaSimplified ( ) [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 reveived formula, if the Boolean is true.

Reimplemented from smtrat::Module.

15.350.3.57 hasLemmas()

`bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

15.350.3.58 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).

15.350.3.59 id()

`std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns A unique ID to identify this module instance.

15.350.3.60 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.

15.350.3.61 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.350.3.62 informBackends()

```
void smtrat::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.

15.350.3.63 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, and smtrat::ICPModule< ICPSettings1 >.

15.350.3.64 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.350.3.65 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::FPPModule< Settings >.

15.350.3.66 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.350.3.67 isLemmaLevel() `bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.350.3.68 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().

15.350.3.69 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.

15.350.3.70 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.350.3.71 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.350.3.72 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.

15.350.3.73 moduleName() `virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]`

Returns The name of the given module type as name.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::EQPreprocessingModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::VSModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::GBModule< Settings >, smtrat::IntEqModule< Setti smtrat::CubeLIAModule< Settings >, smtrat::CurryModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IncWidthModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::BEModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::SplitSOSModule< Settings >.

15.350.3.74 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

15.350.3.75 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

15.350.3.76 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula. TODO: disable this method

15.350.3.77 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula.

15.350.3.78 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to the backends of this module.

15.350.3.79 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to this module.

15.350.3.80 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.350.3.81 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

_out	The stream to print the assignment on.

15.350.3.82 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.350.3.83 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

_out	The stream to print the assignment on.

15.350.3.84 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.350.3.85 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.350.3.86 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.

15.350.3.87 receivedFormulaChecked()

```
void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]
```

Notifies that the received formulas has been checked.

15.350.3.88 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩InfeasibleSubset (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

15.350.3.89 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const [inline], [inherited]
```

15.350.3.90 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

_subformula	The sub formula of the received formula to remove.

Reimplemented from smtrat::Module.

15.350.3.91 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

formula	The sub formula of the received formula to remove.

Reimplemented in smtrat::SATModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::VSModule< Settings >, smtrat::PFEModule< Settings >, smtrat::EQModule< Setting smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Sett smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, and smtrat::GBModule< Settings >.

15.350.3.92 removeOrigin()

`std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin ( ModuleInput::iterator _formula, const FormulaT & _origin ) [inline], [protected], [inherited]`

15.350.3.93 removeOrigins()

`std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩ Origins ( ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]`

15.350.3.94 rPassedFormula()

`const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]`

Returns A reference to the formula passed to the backends of this module.

15.350.3.95 rReceivedFormula()

`const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]`

Returns A reference to the formula passed to this module.

15.350.3.96 runBackends() [1/2] `virtual Answer smtrat::PModule::runBackends ( ) [inline], [virtual]`

Reimplemented from smtrat::Module.

15.350.3.97 runBackends() [2/2]

```
Answer smtrat::PModule::runBackends (
    bool _final,
    bool _full,
    carl::Variable _objective ) [virtual]
```

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.

15.350.3.98 setId()

```
void smtrat::Module::setId (
    std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.350.3.99 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.

15.350.3.100 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.

15.350.3.101 splitUnequalConstraint()

`void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.350.3.102 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.350.3.103 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 >.

15.350.3.104 updateLemmas()

`void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.350.3.105 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.

15.350.3.106 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).

15.350.4 Field Documentation

15.350.4.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.350.4.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.350.4.3 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.

15.350.4.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.350.4.5 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

15.350.4.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.350.4.7 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.

15.350.4.8 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.

15.350.4.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.350.4.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.350.4.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.350.4.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.350.4.13 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.

15.350.4.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.350.4.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.350.4.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.350.4.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.350.4.18 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.

15.350.4.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]`

Reusable splitting variables.

15.350.4.20 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.350.4.21 mSmallerMusesCheckCounter

`unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.350.4.22 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.

15.350.4.23 mTheoryPropagations

`std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]`

15.350.4.24 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).

15.350.4.25 mVariableCounters

`std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.351 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

15.351.1 Member Function Documentation

15.351.1.1 hash_on_level()

```
std::size_t smtrat::cadcells::operators::properties::poly_irreducible↩
_sgn_inv::hash_on_level ( ) const [inline]
```

15.351.1.2 level()

```
size_t smtrat::cadcells::operators::properties::poly_irreducible_sgn_inv↩
::level ( ) const [inline]
```

15.351.2 Field Documentation

15.351.2.1 is_flag

```
constexpr bool smtrat::cadcells::operators::properties::poly_irreducible_↩
sgn_inv::is_flag = false [static], [constexpr]
```

15.351.2.2 poly

```
datastructures::PolyRef smtrat::cadcells::operators::properties::poly_irreducible↩
_sgn_inv::poly
```

15.352 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

15.352.1 Member Function Documentation

15.352.1.1 hash_on_level()

```
std::size_t smtrat::cadcells::operators::properties::poly_ord_inv↩
::hash_on_level ( ) const [inline]
```

15.352.1.2 level()

```
size_t smtrat::cadcells::operators::properties::poly_ord_inv::level ( ) const
[inline]
```

15.352.2 Field Documentation

15.352.2.1 is_flag `constexpr bool smtrat::cadcells::operators::properties::poly_ord_inv::is_flag = false [static], [constexpr]`

15.352.2.2 poly `datastructures::PolyRef smtrat::cadcells::operators::properties::poly_ord_inv↩ ::poly`

15.353 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

15.353.1 Member Function Documentation

15.353.1.1 hash_on_level() `std::size_t smtrat::cadcells::operators::properties::poly_pdel↩ ::hash_on_level ( ) const [inline]`

15.353.1.2 level() `size_t smtrat::cadcells::operators::properties::poly_pdel::level ( ) const [inline]`

15.353.2 Field Documentation

15.353.2.1 is_flag `constexpr bool smtrat::cadcells::operators::properties::poly_pdel::is_flag = false [static], [constexpr]`

15.353.2.2 poly `datastructures::PolyRef smtrat::cadcells::operators::properties::poly_pdel↩ ::poly`

15.354 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

15.354.1 Member Function Documentation

15.354.1.1 hash_on_level() `std::size_t smtrat::cadcells::operators::properties::poly_sgn_inv↩ ::hash_on_level ( ) const [inline]`

15.354.1.2 level() `size_t smtrat::cadcells::operators::properties::poly_sgn_inv::level ( ) const [inline]`

15.354.2 Field Documentation

15.354.2.1 is_flag `constexpr bool smtrat::cadcells::operators::properties::poly_sgn_inv::is_flag = false [static], [constexpr]`

15.354.2.2 poly `datastructures::PolyRef smtrat::cadcells::operators::properties::poly_sgn_inv↩ ::poly`

15.355 smtrat::cad::ProjectionPolynomialInformation::PolyInfo Struct Reference

`#include <ProjectionInformation.h>`

Data Fields

- Origin origin

  Origins of this polynomial.

15.355.1 Field Documentation

15.355.1.1 origin `Origin smtrat::cad::ProjectionPolynomialInformation::PolyInfo::origin`

Origins of this polynomial.

15.356 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

15.356.1 Constructor & Destructor Documentation

15.356.1.1 PolynomialComparator()

```
template<typename PolynomialGetter>
smtrat::cad::PolynomialComparator< PolynomialGetter >::PolynomialComparator (
            const PolynomialGetter * pg,
            std::size_t level ) [inline]
```

15.356.2 Member Function Documentation

15.356.2.1 operator()()

```
template<typename PolynomialGetter>
bool smtrat::cad::PolynomialComparator< PolynomialGetter >::operator() (
            std::size_t lhs,
            std::size_t rhs ) const [inline]
```

15.357 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)

15.357.1 Constructor & Destructor Documentation

15.357.1.1 PolynomialLiftingQueue()

```
template<typename PolynomialGetter>
smtrat::cad::PolynomialLiftingQueue< PolynomialGetter >::PolynomialLiftingQueue (
            const PolynomialGetter * pg,
            std::size_t level )  [inline]
```

15.357.2 Member Function Documentation

15.357.2.1 begin()

```
template<typename PolynomialGetter>
auto smtrat::cad::PolynomialLiftingQueue< PolynomialGetter >::begin ( ) const  [inline]
```

15.357.2.2 disable()

```
template<typename PolynomialGetter>
void smtrat::cad::PolynomialLiftingQueue< PolynomialGetter >::disable (
            std::size_t id )  [inline]
```

15.357.2.3 end()

```
template<typename PolynomialGetter>
auto smtrat::cad::PolynomialLiftingQueue< PolynomialGetter >::end ( ) const  [inline]
```

15.357.2.4 erase()

```
template<typename PolynomialGetter>
auto smtrat::cad::PolynomialLiftingQueue< PolynomialGetter >::erase (
            std::size_t id )  [inline]
```

15.357.2.5 insert()

```
template<typename PolynomialGetter>
auto smtrat::cad::PolynomialLiftingQueue< PolynomialGetter >::insert (
            std::size_t id )  [inline]
```

15.357.2.6 restore()

```
template<typename PolynomialGetter>
void smtrat::cad::PolynomialLiftingQueue< PolynomialGetter >::restore (
            std::size_t id )  [inline]
```

15.357.2.7 size()

```
template<typename PolynomialGetter>
auto smtrat::cad::PolynomialLiftingQueue< PolynomialGetter >::size ( ) const  [inline]
```

15.357.3 Friends And Related Function Documentation

15.357.3.1 operator<<

```
template<typename PolynomialGetter>
template<typename PG >
std::ostream& operator<< (
            std::ostream & os,
            const PolynomialLiftingQueue< PG > & plq )  [friend]
```

15.358 smtrat::cadcells::datastructures::PolyPool Class Reference

A pool for polynomials.

```
#include <polynomials.h>
```

Public Member Functions

- PolyPool (const VariableOrdering &var_order)

  Constructs a polynomial pool with respect to a variable ordering.
- const VariableOrdering & var_order () const
- PolyRef insert (const Poly &poly)
- PolyRef operator() (const Poly &poly)
- const Poly & get (const PolyRef &ref) const
- const Poly & operator() (const PolyRef &ref) const
- bool known (const Poly &poly) const
- void clear_levels (size_t level)

15.358.1 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.

15.358.2 Constructor & Destructor Documentation

15.358.2.1 PolyPool()

```
smtrat::cadcells::datastructures::PolyPool::PolyPool (
            const VariableOrdering & var_order )  [inline]
```

Constructs a polynomial pool with respect to a variable ordering.

Parameters

var_order	The variable ordering determining polynomial levels.

15.358.3 Member Function Documentation

15.358.3.1 clear_levels()

```
void smtrat::cadcells::datastructures::PolyPool::clear_levels (
            size_t level )  [inline]
```

15.358.3.2 get()

```
const Poly& smtrat::cadcells::datastructures::PolyPool::get (
    const PolyRef & ref ) const [inline]
```

15.358.3.3 insert()

```
PolyRef smtrat::cadcells::datastructures::PolyPool::insert (
    const Poly & poly ) [inline]
```

15.358.3.4 known()

```
bool smtrat::cadcells::datastructures::PolyPool::known (
    const Poly & poly ) const [inline]
```

15.358.3.5 operator()() [1/2]

```
PolyRef smtrat::cadcells::datastructures::PolyPool::operator() (
    const Poly & poly ) [inline]
```

15.358.3.6 operator()() [2/2]

```
const Poly& smtrat::cadcells::datastructures::PolyPool::operator()
(
    const PolyRef & ref ) const [inline]
```

15.358.3.7 var_order()

```
const VariableOrdering& smtrat::cadcells::datastructures::PolyPool::var↩
_order ( ) const [inline]
```

15.359 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

15.359.1 Field Documentation

15.359.1.1 disc

```
std::optional<PolyRef> smtrat::cadcells::datastructures::detail::PolyProperties↩
::disc
```

15.359.1.2 factors_nonconst

```
std::vector<PolyRef> smtrat::cadcells::datastructures::detail::↩
PolyProperties::factors_nonconst
```

15.359.1.3 ldcf

```
std::optional<PolyRef> smtrat::cadcells::datastructures::detail::PolyProperties↩
::ldcf
```

15.359.1.4 res

```
std::map<PolyRef, PolyRef> smtrat::cadcells::datastructures::detail::Poly↩
Properties::res
```

15.360 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.

15.360.1 Detailed Description

Refers to a polynomial.

15.360.2 Field Documentation

15.360.2.1 id `size_t smtrat::cadcells::datastructures::PolyRef::id`

The id of the polynomial with respect to its level.

15.360.2.2 level `size_t smtrat::cadcells::datastructures::PolyRef::level`

The level of the polynomial.

15.361 smtrat::PolyTree Class Reference

```
#include <PolyTree.h>
```

Public Types

- enum Type : unsigned { Type::VARIABLE, Type::CONSTANT, Type::SUM, Type::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

15.361.1 Member Enumeration Documentation

15.361.1.1 Type `enum smtrat::PolyTree::Type : unsigned [strong]`

Enumerator

VARIABLE	
CONSTANT	
SUM	
PRODUCT	

15.361.2 Constructor & Destructor Documentation

15.361.2.1 PolyTree()

```
smtrat::PolyTree::PolyTree (
    const Poly & _poly )
```

15.361.3 Member Function Documentation

15.361.3.1 constant()

```
const Integer & smtrat::PolyTree::constant ( ) const
```

15.361.3.2 left()

```
const PolyTree & smtrat::PolyTree::left ( ) const
```

15.361.3.3 poly()

```
const Poly & smtrat::PolyTree::poly ( ) const
```

15.361.3.4 right()

```
const PolyTree & smtrat::PolyTree::right ( ) const
```

15.361.3.5 type()

```
PolyTree::Type smtrat::PolyTree::type ( ) const
```

15.361.3.6 variable()

```
carl::Variable::Arg smtrat::PolyTree::variable ( ) const
```

15.362 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

15.362.1 Constructor & Destructor Documentation

15.362.1.1 PolyTreeContent() [1/3]

```
smtrat::PolyTreeContent::PolyTreeContent (
    const Poly & _poly,
    PolyTree::Type _type,
    const PolyTree & _left,
    const PolyTree & _right ) [inline]
```

15.362.1.2 PolyTreeContent() [2/3]

```
smtrat::PolyTreeContent::PolyTreeContent (
    carl::Variable::Arg _variable ) [inline]
```

15.362.1.3 PolyTreeContent() [3/3] smtrat::PolyTreeContent::PolyTreeContent (Integer _constant) [inline]

15.362.1.4 ~PolyTreeContent() smtrat::PolyTreeContent::~PolyTreeContent ( ) [inline]

15.362.2 Member Function Documentation

15.362.2.1 operator==() bool smtrat::PolyTreeContent::operator== (const PolyTreeContent & _other) const [inline]

15.362.2.2 poly() const Poly& smtrat::PolyTreeContent::poly ( ) const [inline]

15.362.3 Friends And Related Function Documentation

15.362.3.1 PolyTree friend class PolyTree [friend]

15.362.4 Field Documentation

15.362.4.1 mConstant Integer smtrat::PolyTreeContent::mConstant

15.362.4.2 mVariable carl::Variable smtrat::PolyTreeContent::mVariable

15.363 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 ()

15.363.1 Constructor & Destructor Documentation

15.363.1.1 PolyTreePool() smtrat::PolyTreePool::PolyTreePool ( ) [inline], [protected]

Constructor of the pool.

15.363.1.2 ~PolyTreePool() smtrat::PolyTreePool::~PolyTreePool ( ) [inline], [protected]

15.363.2 Member Function Documentation

15.363.2.1 get() const PolyTreeContent * smtrat::PolyTreePool::get (const Poly & _pol)

15.364 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)

15.364.1 Constructor & Destructor Documentation

15.364.1.1 Preprocessor()

```
smtrat::cad::Preprocessor::Preprocessor (
    const std::vector< carl::Variable > & vars ) [inline]
```

15.364.2 Member Function Documentation

15.364.2.1 addConstraint()

```
void smtrat::cad::Preprocessor::addConstraint (
    const ConstraintT & c )
```

15.364.2.2 clear()

```
void smtrat::cad::Preprocessor::clear ( ) [inline]
```

15.364.2.3 getConflict()

```
const auto& smtrat::cad::Preprocessor::getConflict ( ) const [inline]
```

15.364.2.4 model()

```
const Model& smtrat::cad::Preprocessor::model ( ) const [inline]
```

15.364.2.5 postprocessConflict()

```
void smtrat::cad::Preprocessor::postprocessConflict (
    std::set< FormulaT > & mis ) const
```

15.364.2.6 preprocess()

```
bool smtrat::cad::Preprocessor::preprocess ( )
```

15.364.2.7 removeConstraint()

```
void smtrat::cad::Preprocessor::removeConstraint (
    const ConstraintT & c )
```

15.364.2.8 result()

```
template<typename Map >
preprocessor::ConstraintUpdate smtrat::cad::Preprocessor::result (
    const Map & oldC ) const [inline]
```

15.364.2.9 simplify()

```
FormulaT smtrat::cad::Preprocessor::simplify (
    const FormulaT & f ) const [inline]
```

15.364.3 Friends And Related Function Documentation

15.364.3.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const Preprocessor & cadpp ) [friend]
```

15.365 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

15.365.1 Member Function Documentation

15.365.1.1 register_hook()

```
static bool smtrat::cad::PreprocessorSettings::register_hook ( ) [inline], [static]
```

15.365.1.2 register_settings()

```
static void smtrat::cad::PreprocessorSettings::register_settings (
    SettingsParser & parser ) [inline], [static]
```

15.365.2 Field Documentation

15.365.2.1 disable_resultants

```
bool smtrat::cad::PreprocessorSettings::disable_resultants = false
```

15.365.2.2 disable_variable_elimination

```
bool smtrat::cad::PreprocessorSettings::disable_variable↩
_elimination = false
```

15.365.2.3 dummy

```
const bool smtrat::cad::PreprocessorSettings::dummy [static]
```

15.366 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.

15.366.1 Field Documentation

15.366.1.1 rwth_slurm `bool benchmax::settings::PresetSettings::rwth_slurm`

Use slurm on the RWTH HPC cluster.

15.366.1.2 rwth_slurm_name `std::string benchmax::settings::PresetSettings::rwth_slurm_name`

Name for this job.

15.367 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 removeIf (F &&f)
- void clear ()

Data Fields

- T **elements**

  STL member.

15.367.1 Constructor & Destructor Documentation

15.367.1.1 PriorityQueue() [1/2]

```
template<typename T, typename Compare = std::less<T>>
smtrat::PriorityQueue< T, Compare >::PriorityQueue ( ) [inline], [explicit]
```

15.367.1.2 PriorityQueue() [2/2]

```
template<typename T, typename Compare = std::less<T>>
smtrat::PriorityQueue< T, Compare >::PriorityQueue (
            const Compare & comp ) [inline], [explicit]
```

15.367.2 Member Function Documentation

15.367.2.1 begin() [1/2]

```
template<typename T, typename Compare = std::less<T>>
auto smtrat::PriorityQueue< T, Compare >::begin ( ) [inline]
```

15.367.2.2 begin() [2/2]

```
template<typename T, typename Compare = std::less<T>>
auto smtrat::PriorityQueue< T, Compare >::begin ( ) const [inline]
```

15.367.2.3 clear()

```
template<typename T, typename Compare = std::less<T>>
void smtrat::PriorityQueue< T, Compare >::clear ( ) [inline]
```

15.367.2.4 data() [1/2]

```
template<typename T, typename Compare = std::less<T>>
auto& smtrat::PriorityQueue< T, Compare >::data ( ) [inline]
```

15.367.2.5 data() [2/2]

```
template<typename T, typename Compare = std::less<T>>
const auto& smtrat::PriorityQueue< T, Compare >::data ( ) const [inline]
```

15.367.2.6 end() [1/2]

```
template<typename T, typename Compare = std::less<T>>
auto smtrat::PriorityQueue< T, Compare >::end ( ) [inline]
```

15.367.2.7 end() [2/2]

```
template<typename T, typename Compare = std::less<T>>
auto smtrat::PriorityQueue< T, Compare >::end ( ) const [inline]
```

15.367.2.8 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]
```

15.367.2.9 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]
```

15.367.2.10 find()

```
template<typename T, typename Compare = std::less<T>>
auto smtrat::PriorityQueue< T, Compare >::find (
            const T & t ) const [inline]
```

15.367.2.11 fix()

```
template<typename T, typename Compare = std::less<T>>
void smtrat::PriorityQueue< T, Compare >::fix ( ) [inline]
```

15.367.2.12 removeIf()

```
template<typename T, typename Compare = std::less<T>>
template<typename F >
void smtrat::PriorityQueue< T, Compare >::removeIf (
            F && f ) [inline]
```

15.367.3 Field Documentation

15.367.3.1 elements

```
T std::priority_queue< T >::elements [inherited]
```

STL member.

15.368 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.

15.368.1 Detailed Description This class iteratively applies the operators to a smtlib file until no further simplifications can be performed.

15.368.2 Constructor & Destructor Documentation

15.368.2.1 Producer()

```
delta::Producer::Producer (
    const Checker & checker,
    const Settings & settings ) [inline]
```

Create a new manager that uses the given checker.

Parameters

checker	Checker to call the solver.
settings	Settings object.

15.368.3 Member Function Documentation

15.368.3.1 interrupt()

```
void delta::Producer::interrupt ( ) const [inline]
```

15.368.3.2 operator()()

```
std::size_t delta::Producer::operator() (
    Node & root ) [inline]
```

Apply simplifications to the given node.

Parameters

root	Node to simplify.

15.369 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`.

15.369.1 Detailed Description This class provides a simple way to show progress bars on the command line.

15.369.2 Member Function Documentation

15.369.2.1 operator()() [1/4] `void delta::ProgressBar::operator() ( ) [inline]`

Increase progress by one and show progress bar.

15.369.2.2 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

p	Progress.

15.369.2.3 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

progress	Progress.
total	Total.

15.369.2.4 operator()() [4/4] `void delta::ProgressBar::operator() ( std::size_t steps ) [inline]`

Increase progress by number of steps and show progress bar.

Parameters

steps	Steps to move forward.

15.370 smtrat::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
- 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

- 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.

15.370.1 Member Typedef Documentation

15.370.1.1 Constraints

```
template<typename Settings>
using smtrat::cad::BaseProjection< Settings >::Constraints = CADConstraints<Settings::backtracking>
[protected], [inherited]
```

15.370.2 Member Function Documentation

15.370.2.1 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.

15.370.2.2 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 >.

15.370.2.3 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.

15.370.2.4 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::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Incrementality::NONE smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, smtrat::cad::Projection< Incrementality::FULL, BT, S and smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >.

15.370.2.5 callRemoveCallback() template<typename Settings>

```
void smtrat::cad::BaseProjection< Settings >::callRemoveCallback (
            std::size_t level,
            const SampleLiftedWith & slw ) const  [inline], [protected], [inherited]
```

15.370.2.6 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.

15.370.2.7 dim() template<typename Settings>

```
std::size_t smtrat::cad::BaseProjection< Settings >::dim ( ) const  [inline], [inherited]
```

Returns the dimension of the projection.

15.370.2.8 empty() [1/2] template<typename Settings>

```
virtual bool smtrat::cad::BaseProjection< Settings >::empty ( )  [inline], [virtual], [inherited]
```

15.370.2.9 empty() [2/2] template<typename Settings>

```
virtual bool smtrat::cad::BaseProjection< Settings >::empty (
            std::size_t level ) const  [pure virtual], [inherited]
```

Implemented in smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, smtrat::cad::Projection< Incremental smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Increme smtrat::qe::cad::Projection< Settings >, smtrat::qe::cad::Projection< smtrat::qe::cad::CADSettings >, and smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >.

15.370.2.10 exportAsDot() template<typename Settings>

```
virtual void smtrat::cad::BaseProjection< Settings >::exportAsDot (
            std::ostream &  ) const  [inline], [virtual], [inherited]
```

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, and smtrat::cad::Projection< Incre

15.370.2.11 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.

15.370.2.12 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.

15.370.2.13 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 >.

15.370.2.14 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.

15.370.2.15 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::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, smtrat::cad::Projection< Incremental smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Increme smtrat::qe::cad::Projection< Settings >, smtrat::qe::cad::Projection< smtrat::qe::cad::CADSettings >, and smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >.

15.370.2.16 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::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, smtrat::cad::Projection< Incremental smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Increme smtrat::qe::cad::Projection< Settings >, smtrat::qe::cad::Projection< smtrat::qe::cad::CADSettings >, and smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >.

15.370.2.17 isPurged() template<typename Settings>

```
bool smtrat::cad::BaseProjection< Settings >::isPurged (
            std::size_t level,
            std::size_t id )  [inline], [protected], [inherited]
```

15.370.2.18 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.

15.370.2.19 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::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Incrementality::FULL smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >, and smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >.

15.370.2.20 reset() template<typename Settings>

```
void smtrat::cad::BaseProjection< Settings >::reset ( )  [inline], [inherited]
```

Resets all datastructures, use the given variables from now on.

15.370.2.21 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.

15.370.2.22 size() [1/2] template<typename Settings>

```
std::size_t smtrat::cad::BaseProjection< Settings >::size ( ) const  [inline], [inherited]
```

15.370.2.23 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::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, smtrat::cad::Projection< Incremental smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Increme smtrat::qe::cad::Projection< Settings >, smtrat::qe::cad::Projection< smtrat::qe::cad::CADSettings >, and smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >.

15.370.2.24 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.

15.370.2.25 vars() template<typename Settings>

```
const auto& smtrat::cad::BaseProjection< Settings >::vars ( ) const [inline], [inherited]
```

Returns the variables used for projection.

15.370.3 Field Documentation

15.370.3.1 mConstraints template<typename Settings>

```
const Constraints& smtrat::cad::BaseProjection< Settings >::mConstraints [protected], [inherited]
```

15.370.3.2 mInfo template<typename Settings>

```
ProjectionInformation smtrat::cad::BaseProjection< Settings >::mInfo [protected], [inherited]
```

Additional info on projection, projection levels and projection polynomials.

15.370.3.3 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.

15.370.3.4 mOperator template<typename Settings>

```
ProjectionOperator smtrat::cad::BaseProjection< Settings >::mOperator [protected], [inherited]
```

The projection operator.

15.370.3.5 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.

15.371 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)

15.371.1 Constructor & Destructor Documentation

15.371.1.1 Projection()

```
template<typename Settings>
smtrat::qe::cad::Projection< Settings >::Projection (
            const Constraints & c )  [inline]
```

15.371.2 Member Function Documentation

15.371.2.1 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.

15.371.2.2 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 >.

15.371.2.3 addPolynomial() [1/3]

```
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.

15.371.2.4 addPolynomial() [2/3] 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::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Incrementality::NONE smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, smtrat::cad::Projection< Incrementality::FULL, BT, S and smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >.

15.371.2.5 addPolynomial() [3/3] template<typename Settings>

```
carl::Bitset smtrat::qe::cad::Projection< Settings >::addPolynomial (
            const UPoly & p,
            std::size_t cid,
            bool  )  [inline], [override]
```

15.371.2.6 callRemoveCallback() template<typename Settings>

```
void smtrat::cad::BaseProjection< Settings >::callRemoveCallback (
            std::size_t level,
            const SampleLiftedWith & slw ) const  [inline], [protected], [inherited]
```

15.371.2.7 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.

15.371.2.8 dim() [1/2] template<typename Settings>

```
std::size_t smtrat::cad::BaseProjection< Settings >::dim  [inline]
```

Returns the dimension of the projection.

15.371.2.9 dim() [2/2] template<typename Settings>

```
std::size_t smtrat::cad::BaseProjection< Settings >::dim ( ) const  [inline], [inherited]
```

Returns the dimension of the projection.

15.371.2.10 empty() [1/2] template<typename Settings>

```
virtual bool smtrat::cad::BaseProjection< Settings >::empty ( )  [inline], [virtual], [inherited]
```

15.371.2.11 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 >.

15.371.2.12 exportAsDot() template<typename Settings>

```
virtual void smtrat::cad::BaseProjection< Settings >::exportAsDot (
            std::ostream & ) const [inline], [virtual], [inherited]
```

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, and smtrat::cad::Projection< Incre

15.371.2.13 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.

15.371.2.14 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.

15.371.2.15 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 >.

15.371.2.16 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.

15.371.2.17 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 >.

15.371.2.18 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 >.

15.371.2.19 isPurged() template<typename Settings>

```
bool smtrat::cad::BaseProjection< Settings >::isPurged (
            std::size_t level,
            std::size_t id ) [inline], [protected], [inherited]
```

15.371.2.20 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.

15.371.2.21 removePolynomial() [2/3] template<typename Settings>

```
void smtrat::qe::cad::Projection< Settings >::removePolynomial (
            const UPoly & ,
            std::size_t cid,
            bool ) [inline], [override]
```

15.371.2.22 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::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Incrementality::FULL smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >, and smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >.

15.371.2.23 removeProjectionFactor() template<typename Settings>

```
void smtrat::qe::cad::Projection< Settings >::removeProjectionFactor (
            std::size_t level,
            std::size_t id ) [inline]
```

15.371.2.24 reset() template<typename Settings>

```
void smtrat::qe::cad::Projection< Settings >::reset ( ) [inline]
```

15.371.2.25 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.

15.371.2.26 size() [1/4] template<typename Settings>

```
std::size_t smtrat::cad::BaseProjection< Settings >::size [inline]
```

15.371.2.27 size() [2/4] template<typename Settings>

```
std::size_t smtrat::cad::BaseProjection< Settings >::size ( ) const [inline], [inherited]
```

15.371.2.28 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 >.

15.371.2.29 size() [4/4] template<typename Settings>

```
virtual std::size_t smtrat::cad::BaseProjection< Settings >::size
```

15.371.2.30 testProjectionFactor() template<typename Settings>

```
bool smtrat::qe::cad::Projection< Settings >::testProjectionFactor (
            std::size_t level,
            std::size_t id ) const [inline]
```

15.371.2.31 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.

15.371.2.32 vars() template<typename Settings>

```
const auto& smtrat::cad::BaseProjection< Settings >::vars ( ) const [inline], [inherited]
```

Returns the variables used for projection.

15.371.3 Friends And Related Function Documentation

15.371.3.1 operator<< template<typename Settings>

```
template<typename S >
std::ostream& operator<< (
            std::ostream & os,
            const Projection< S > & p ) [friend]
```

15.371.4 Field Documentation

15.371.4.1 mConstraints template<typename Settings>

`const Constraints& smtrat::cad::BaseProjection< Settings >::mConstraints [protected], [inherited]`

15.371.4.2 mInfo template<typename Settings>

`ProjectionInformation smtrat::cad::BaseProjection< Settings >::mInfo [protected], [inherited]`

Additional info on projection, projection levels and projection polynomials.

15.371.4.3 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.

15.372 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)

15.372.1 Constructor & Destructor Documentation

15.372.1.1 Projection() `template<typename Settings >`

```
smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >::Projection (
            const Constraints & c )  [inline]
```

15.372.2 Member Function Documentation

15.372.2.1 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.

15.372.2.2 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 >.

15.372.2.3 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.

15.372.2.4 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 >.

15.372.2.5 callRemoveCallback() template<typename Settings>

```
void smtrat::cad::BaseProjection< Settings >::callRemoveCallback (
            std::size_t level,
            const SampleLiftedWith & slw ) const  [inline], [protected], [inherited]
```

15.372.2.6 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.

15.372.2.7 dim() [1/2] template<typename Settings >

```
std::size_t smtrat::cad::BaseProjection< Settings >::dim  [inline]
```

Returns the dimension of the projection.

15.372.2.8 dim() [2/2] template<typename Settings>

```
std::size_t smtrat::cad::BaseProjection< Settings >::dim ( ) const [inline], [inherited]
```

Returns the dimension of the projection.

15.372.2.9 empty() [1/2] template<typename Settings>

```
virtual bool smtrat::cad::BaseProjection< Settings >::empty ( ) [inline], [virtual], [inherited]
```

15.372.2.10 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 >.

15.372.2.11 exportAsDot() template<typename Settings >

```
void smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >::exportAs↩
Dot (
            std::ostream & out ) const [inline], [override], [virtual]
```

Reimplemented from smtrat::cad::BaseProjection< Settings >.

15.372.2.12 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.

15.372.2.13 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.

15.372.2.14 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 >.

15.372.2.15 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.

15.372.2.16 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 >.

15.372.2.17 hasPolynomialById() template<typename Settings >

```
bool smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >::has↩
PolynomialById (
            std::size_t level,
            std::size_t id ) const  [inline], [override], [virtual]
```

Implements smtrat::cad::BaseProjection< Settings >.

15.372.2.18 isPurged() template<typename Settings>

```
bool smtrat::cad::BaseProjection< Settings >::isPurged (
            std::size_t level,
            std::size_t id )  [inline], [protected], [inherited]
```

15.372.2.19 printPolynomialIDs() template<typename Settings >

```
void smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >::print↩
PolynomialIDs ( ) const  [inline]
```

15.372.2.20 projectNewPolynomial() template<typename Settings >

```
carl::Bitset smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >↩
::projectNewPolynomial (
            const ConstraintSelection & cs = carl::Bitset(true) )  [inline]
```

15.372.2.21 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.

15.372.2.22 removePolynomial() [2/2]

```
template<typename Settings >
void smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >::remove↩
Polynomial (
            const UPoly & p,
            std::size_t cid,
            bool isBound )  [inline], [override], [virtual]
```

Removes the given polynomial from the projection.

Implements smtrat::cad::BaseProjection< Settings >.

15.372.2.23 reset()

```
template<typename Settings >
void smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >::reset ( )
[inline]
```

15.372.2.24 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.

15.372.2.25 size() [1/4]

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::size  [inline]
```

15.372.2.26 size() [2/4]

```
template<typename Settings>
std::size_t smtrat::cad::BaseProjection< Settings >::size ( ) const  [inline], [inherited]
```

15.372.2.27 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 >.

15.372.2.28 size() [4/4]

```
template<typename Settings >
virtual std::size_t smtrat::cad::BaseProjection< Settings >::size
```

15.372.2.29 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.

15.372.2.30 vars() template<typename Settings>

```
const auto& smtrat::cad::BaseProjection< Settings >::vars ( ) const [inline], [inherited]
```

Returns the variables used for projection.

15.372.3 Friends And Related Function Documentation

15.372.3.1 operator<< template<typename Settings >

```
template<typename S >
std::ostream& operator<< (
            std::ostream & os,
            const Projection< Incrementality::FULL, Backtracking::HIDE, S > & p ) [friend]
```

15.372.4 Field Documentation

15.372.4.1 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.

15.373 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)

15.373.1 Constructor & Destructor Documentation

15.373.1.1 Projection()

```
template<typename Settings , Backtracking BT>
smtrat::cad::Projection< Incrementality::FULL, BT, Settings >::Projection (
            const Constraints & c )  [inline]
```

15.373.2 Member Function Documentation

15.373.2.1 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.

15.373.2.2 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 >.

15.373.2.3 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.

15.373.2.4 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 >.

15.373.2.5 callRemoveCallback()

```
template<typename Settings>
void smtrat::cad::BaseProjection< Settings >::callRemoveCallback (
            std::size_t level,
            const SampleLiftedWith & slw ) const [inline], [protected], [inherited]
```

15.373.2.6 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.

15.373.2.7 dim() [1/2]

```
template<typename Settings , Backtracking BT>
std::size_t smtrat::cad::BaseProjection< Settings >::dim [inline]
```

Returns the dimension of the projection.

15.373.2.8 dim() [2/2] template<typename Settings>

```
std::size_t smtrat::cad::BaseProjection< Settings >::dim ( ) const [inline], [inherited]
```

Returns the dimension of the projection.

15.373.2.9 empty() [1/2] template<typename Settings>

```
virtual bool smtrat::cad::BaseProjection< Settings >::empty ( ) [inline], [virtual], [inherited]
```

15.373.2.10 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 >.

15.373.2.11 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 >.

15.373.2.12 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.

15.373.2.13 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.

15.373.2.14 getOrigin() template<typename Settings , Backtracking BT>

```
Origin smtrat::cad::Projection< Incrementality::FULL, BT, Settings >::getOrigin (
            std::size_t level,
            std::size_t id ) const [inline], [override], [virtual]
```

Reimplemented from smtrat::cad::BaseProjection< Settings >.

15.373.2.15 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.

15.373.2.16 getPolynomialById()

```
template<typename Settings , Backtracking BT>
const UPoly& smtrat::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 smtrat::cad::BaseProjection< Settings >.

15.373.2.17 hasPolynomialById()

```
template<typename Settings , Backtracking BT>
bool smtrat::cad::Projection< Incrementality::FULL, BT, Settings >::hasPolynomialById (
            std::size_t level,
            std::size_t id ) const  [inline], [override], [virtual]
```

Implements smtrat::cad::BaseProjection< Settings >.

15.373.2.18 isPurged()

```
template<typename Settings>
bool smtrat::cad::BaseProjection< Settings >::isPurged (
            std::size_t level,
            std::size_t id )  [inline], [protected], [inherited]
```

15.373.2.19 projectNewPolynomial()

```
template<typename Settings , Backtracking BT>
carl::Bitset smtrat::cad::Projection< Incrementality::FULL, BT, Settings >::projectNewPolynomial
(
            const ConstraintSelection & cs = carl::Bitset(true) )  [inline]
```

15.373.2.20 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.

15.373.2.21 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 >.

15.373.2.22 reset()

```
template<typename Settings , Backtracking BT>
void smtrat::cad::Projection< Incrementality::FULL, BT, Settings >::reset ( ) [inline]
```

15.373.2.23 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.

15.373.2.24 size() [1/4]

```
template<typename Settings , Backtracking BT>
std::size_t smtrat::cad::BaseProjection< Settings >::size [inline]
```

15.373.2.25 size() [2/4]

```
template<typename Settings>
std::size_t smtrat::cad::BaseProjection< Settings >::size ( ) const [inline], [inherited]
```

15.373.2.26 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 >.

15.373.2.27 size() [4/4]

```
template<typename Settings , Backtracking BT>
virtual std::size_t smtrat::cad::BaseProjection< Settings >::size
```

15.373.2.28 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.

15.373.2.29 vars()

```
template<typename Settings>
const auto& smtrat::cad::BaseProjection< Settings >::vars ( ) const [inline], [inherited]
```

Returns the variables used for projection.

15.373.3 Friends And Related Function Documentation

15.373.3.1 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]
```

15.373.4 Field Documentation

15.373.4.1 mInfo `template<typename Settings>`

`ProjectionInformation smtrat::cad::BaseProjection< Settings >::mInfo [protected], [inherited]`

Additional info on projection, projection levels and projection polynomials.

15.373.4.2 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.

15.374 smtrat::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)

15.374.1 Detailed Description

template<typename Settings>
class smtrat::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.

15.374.2 Constructor & Destructor Documentation

15.374.2.1 Projection()

```
template<typename Settings >
smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >::Projection (
            const Constraints & c )  [inline]
```

15.374.3 Member Function Documentation

15.374.3.1 addEqConstraint() [1/2]

```
template<typename Settings>
carl::Bitset smtrat::cad::BaseProjection< Settings >::addEqConstraint (
            const Poly & p,
            std::size_t cid,
            bool isBound )  [inline], [inherited]
```

Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.

15.374.3.2 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], [inherited]
```

Adds the given polynomial of an equational constraint to the projection.

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >.

15.374.3.3 addPolynomial() [1/2]

```
template<typename Settings>
carl::Bitset smtrat::cad::BaseProjection< Settings >::addPolynomial (
            const Poly & p,
            std::size_t cid,
            bool isBound )  [inline], [inherited]
```

Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.

15.374.3.4 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 >.

15.374.3.5 callRemoveCallback() template<typename Settings>

```
void smtrat::cad::BaseProjection< Settings >::callRemoveCallback (
            std::size_t level,
            const SampleLiftedWith & slw ) const [inline], [protected], [inherited]
```

15.374.3.6 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.

15.374.3.7 dim() [1/2] template<typename Settings >

```
std::size_t smtrat::cad::BaseProjection< Settings >::dim [inline]
```

Returns the dimension of the projection.

15.374.3.8 dim() [2/2] template<typename Settings>

```
std::size_t smtrat::cad::BaseProjection< Settings >::dim ( ) const [inline], [inherited]
```

Returns the dimension of the projection.

15.374.3.9 empty() [1/2] template<typename Settings>

```
virtual bool smtrat::cad::BaseProjection< Settings >::empty ( ) [inline], [virtual], [inherited]
```

15.374.3.10 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 >.

15.374.3.11 exportAsDot() template<typename Settings>

```
virtual void smtrat::cad::BaseProjection< Settings >::exportAsDot (
            std::ostream & ) const [inline], [virtual], [inherited]
```

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, and smtrat::cad::Projection< Incre

15.374.3.12 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.

15.374.3.13 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.

15.374.3.14 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 >.

15.374.3.15 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.

15.374.3.16 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 >.

15.374.3.17 hasPolynomialById() template<typename Settings >

```
bool smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >::has↩
PolynomialById (
            std::size_t level,
            std::size_t id ) const [inline], [override], [virtual]
```

Implements smtrat::cad::BaseProjection< Settings >.

15.374.3.18 isPurged() template<typename Settings>

```
bool smtrat::cad::BaseProjection< Settings >::isPurged (
            std::size_t level,
            std::size_t id ) [inline], [protected], [inherited]
```

15.374.3.19 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.

15.374.3.20 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.

15.374.3.21 removePolynomial() [2/2]

```
template<typename Settings >
void smtrat::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 smtrat::cad::BaseProjection< Settings >.

15.374.3.22 reset()

```
template<typename Settings >
void smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >::reset (
)  [inline]
```

Resets all datastructures, use the given variables from now on.

15.374.3.23 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.

15.374.3.24 size() [1/4]

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::size  [inline]
```

15.374.3.25 size() [2/4] template<typename Settings>

```
std::size_t smtrat::cad::BaseProjection< Settings >::size ( ) const [inline], [inherited]
```

15.374.3.26 size() [3/4] template<typename Settings >

```
std::size_t smtrat::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 smtrat::cad::BaseProjection< Settings >.

15.374.3.27 size() [4/4] template<typename Settings >

```
virtual std::size_t smtrat::cad::BaseProjection< Settings >::size
```

15.374.3.28 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.

15.374.3.29 vars() template<typename Settings>

```
const auto& smtrat::cad::BaseProjection< Settings >::vars ( ) const [inline], [inherited]
```

Returns the variables used for projection.

15.374.4 Friends And Related Function Documentation

15.374.4.1 operator<< template<typename Settings >

```
template<typename S >
std::ostream& operator<< (
            std::ostream & os,
            const Projection< Incrementality::NONE, Backtracking::ORDERED, S > & p ) [friend]
```

15.374.5 Field Documentation

15.374.5.1 mConstraints template<typename Settings>

```
const Constraints& smtrat::cad::BaseProjection< Settings >::mConstraints [protected], [inherited]
```

15.374.5.2 mInfo template<typename Settings>

```
ProjectionInformation smtrat::cad::BaseProjection< Settings >::mInfo [protected], [inherited]
```

Additional info on projection, projection levels and projection polynomials.

15.374.5.3 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.

15.375 smtrat::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
- 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::UNORDERED, S > &p)

15.375.1 Detailed Description

template<typename Settings>
class smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >

This class implements a projection that supports no incrementality and allows backtracking to be out of order.

15.375.2 Constructor & Destructor Documentation

15.375.2.1 Projection()

```
template<typename Settings >
smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >::Projection
(
            const Constraints & c )  [inline]
```

15.375.3 Member Function Documentation

15.375.3.1 addEqConstraint() [1/2]

```
template<typename Settings>
carl::Bitset smtrat::cad::BaseProjection< Settings >::addEqConstraint (
            const Poly & p,
            std::size_t cid,
            bool isBound )  [inline], [inherited]
```

Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.

15.375.3.2 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], [inherited]
```

Adds the given polynomial of an equational constraint to the projection.

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >.

15.375.3.3 addPolynomial() [1/2] template<typename Settings>

```
carl::Bitset smtrat::cad::BaseProjection< Settings >::addPolynomial (
            const Poly & p,
            std::size_t cid,
            bool isBound ) [inline], [inherited]
```

Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.

15.375.3.4 addPolynomial() [2/2] template<typename Settings >

```
carl::Bitset smtrat::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 smtrat::cad::BaseProjection< Settings >.

Reimplemented in smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >.

15.375.3.5 callRemoveCallback() template<typename Settings>

```
void smtrat::cad::BaseProjection< Settings >::callRemoveCallback (
            std::size_t level,
            const SampleLiftedWith & slw ) const [inline], [protected], [inherited]
```

15.375.3.6 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.

15.375.3.7 dim() [1/2] template<typename Settings >

```
std::size_t smtrat::cad::BaseProjection< Settings >::dim [inline]
```

Returns the dimension of the projection.

15.375.3.8 dim() [2/2] template<typename Settings>

```
std::size_t smtrat::cad::BaseProjection< Settings >::dim ( ) const [inline], [inherited]
```

Returns the dimension of the projection.

15.375.3.9 empty() [1/2] template<typename Settings>

```
virtual bool smtrat::cad::BaseProjection< Settings >::empty ( ) [inline], [virtual], [inherited]
```

15.375.3.10 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 >.

15.375.3.11 exportAsDot() template<typename Settings>

```
virtual void smtrat::cad::BaseProjection< Settings >::exportAsDot (
            std::ostream & ) const [inline], [virtual], [inherited]
```

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, and smtrat::cad::Projection< Incre

15.375.3.12 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.

15.375.3.13 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.

15.375.3.14 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 >.

15.375.3.15 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.

15.375.3.16 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]
```

Retrieves a polynomial from its id.

Implements smtrat::cad::BaseProjection< Settings >.

15.375.3.17 hasPolynomialById()

```
template<typename Settings >
bool smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >::has↩
PolynomialById (
            std::size_t level,
            std::size_t id ) const  [inline], [override], [virtual]
```

Implements smtrat::cad::BaseProjection< Settings >.

15.375.3.18 isPurged()

```
template<typename Settings>
bool smtrat::cad::BaseProjection< Settings >::isPurged (
            std::size_t level,
            std::size_t id )  [inline], [protected], [inherited]
```

15.375.3.19 projectNewPolynomial()

```
template<typename Settings >
carl::Bitset smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings
>::projectNewPolynomial (
            const ConstraintSelection &  = carl::Bitset(true) )  [inline]
```

15.375.3.20 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.

15.375.3.21 removePolynomial() [2/2]

```
template<typename Settings >
void smtrat::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 smtrat::cad::BaseProjection< Settings >.

Reimplemented in smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >.

15.375.3.22 reset()

```
template<typename Settings >
void smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >::reset
( )  [inline]
```

15.375.3.23 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.

15.375.3.24 size() [1/4]

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::size  [inline]
```

15.375.3.25 size() [2/4]

```
template<typename Settings>
std::size_t smtrat::cad::BaseProjection< Settings >::size ( ) const  [inline], [inherited]
```

15.375.3.26 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 >.

15.375.3.27 size() [4/4]

```
template<typename Settings >
virtual std::size_t smtrat::cad::BaseProjection< Settings >::size
```

15.375.3.28 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.

15.375.3.29 vars() template<typename Settings>

```
const auto& smtrat::cad::BaseProjection< Settings >::vars ( ) const  [inline], [inherited]
```

Returns the variables used for projection.

15.375.4 Friends And Related Function Documentation

15.375.4.1 operator<< template<typename Settings >

```
template<typename S >
std::ostream& operator<< (
            std::ostream & os,
            const Projection< Incrementality::NONE, Backtracking::UNORDERED, S > & p )  [friend]
```

15.375.5 Field Documentation

15.375.5.1 mConstraints template<typename Settings>

```
const Constraints& smtrat::cad::BaseProjection< Settings >::mConstraints  [protected], [inherited]
```

15.375.5.2 mInfo template<typename Settings>

```
ProjectionInformation smtrat::cad::BaseProjection< Settings >::mInfo  [protected], [inherited]
```

Additional info on projection, projection levels and projection polynomials.

15.375.5.3 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.

15.376 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)

  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.
- 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)

15.376.1 Constructor & Destructor Documentation

15.376.1.1 Projection()

```
template<typename Settings , Backtracking BT>
template<typename ... Args>
smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >::Projection (
            Args &&... args ) [inline]
```

15.376.2 Member Function Documentation

15.376.2.1 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.

15.376.2.2 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 >.

15.376.2.3 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.

15.376.2.4 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 >.

15.376.2.5 callRemoveCallback() template<typename Settings>

```
void smtrat::cad::BaseProjection< Settings >::callRemoveCallback (
            std::size_t level,
            const SampleLiftedWith & slw ) const  [inline], [protected], [inherited]
```

15.376.2.6 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.

15.376.2.7 dim() [1/2] template<typename Settings >

```
std::size_t smtrat::cad::BaseProjection< Settings >::dim [inline], [inherited]
```

Returns the dimension of the projection.

15.376.2.8 dim() [2/2] template<typename Settings>

```
std::size_t smtrat::cad::BaseProjection< Settings >::dim ( ) const [inline], [inherited]
```

Returns the dimension of the projection.

15.376.2.9 empty() [1/2] template<typename Settings>

```
virtual bool smtrat::cad::BaseProjection< Settings >::empty ( ) [inline], [virtual], [inherited]
```

15.376.2.10 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 >.

15.376.2.11 exportAsDot() template<typename Settings>

```
virtual void smtrat::cad::BaseProjection< Settings >::exportAsDot (
            std::ostream & ) const [inline], [virtual], [inherited]
```

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, and smtrat::cad::Projection< Incre

15.376.2.12 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.

15.376.2.13 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.

15.376.2.14 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 >.

15.376.2.15 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.

15.376.2.16 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 >.

15.376.2.17 hasPolynomialById()

```
template<typename Settings >
bool smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >::has↩
PolynomialById (
            std::size_t level,
            std::size_t id ) const  [inline], [override], [virtual], [inherited]
```

Implements smtrat::cad::BaseProjection< Settings >.

15.376.2.18 isPurged()

```
template<typename Settings>
bool smtrat::cad::BaseProjection< Settings >::isPurged (
            std::size_t level,
            std::size_t id )  [inline], [protected], [inherited]
```

15.376.2.19 projectNewPolynomial()

```
template<typename Settings , Backtracking BT>
carl::Bitset smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >::projectNew↩
Polynomial (
            const ConstraintSelection &  = carl::Bitset(true) )  [inline]
```

15.376.2.20 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.

15.376.2.21 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 >.

15.376.2.22 reset()

```
template<typename Settings , Backtracking BT>
void smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >::reset ( )  [inline]
```

15.376.2.23 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.

15.376.2.24 size() [1/4]

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::size  [inline], [inherited]
```

15.376.2.25 size() [2/4]

```
template<typename Settings>
std::size_t smtrat::cad::BaseProjection< Settings >::size ( ) const  [inline], [inherited]
```

15.376.2.26 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 >.

15.376.2.27 size() [4/4]

```
template<typename Settings >
virtual std::size_t smtrat::cad::BaseProjection< Settings >::size  [inherited]
```

15.376.2.28 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.

15.376.2.29 vars() template<typename Settings>

```
const auto& smtrat::cad::BaseProjection< Settings >::vars ( ) const  [inline], [inherited]
```

Returns the variables used for projection.

15.376.3 Friends And Related Function Documentation

15.376.3.1 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]
```

15.376.4 Field Documentation

15.376.4.1 mConstraints template<typename Settings>

```
const Constraints& smtrat::cad::BaseProjection< Settings >::mConstraints  [protected], [inherited]
```

15.376.4.2 mInfo template<typename Settings>

```
ProjectionInformation smtrat::cad::BaseProjection< Settings >::mInfo  [protected], [inherited]
```

Additional info on projection, projection levels and projection polynomials.

15.376.4.3 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.

15.377 smtrat::cad::projection_compare::ProjectionComparator< Strategy > Struct Template Reference

```
#include <ProjectionComparator.h>
```

15.378 smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::D > Struct Template Reference

```
#include <ProjectionComparator.h>
```

Public Member Functions

- bool operator() (const Candidate< Poly > &lhs, const Candidate< Poly > &rhs) const

15.378.1 Member Function Documentation

15.378.1.1 operator()() bool smtrat::cad::projection_compare::ProjectionComparator_impl< Args >::operator() (

```
            const Candidate< Poly > & lhs,
            const Candidate< Poly > & rhs ) const  [inline], [inherited]
```

15.379 smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::ID > Struct Template Reference

```
#include <ProjectionComparator.h>
```

Public Member Functions

- bool operator() (const Candidate< Poly > &lhs, const Candidate< Poly > &rhs) const

15.379.1 Member Function Documentation

15.379.1.1 operator()()

```
bool smtrat::cad::projection_compare::ProjectionComparator_impl< Args >::operator() (
            const Candidate< Poly > & lhs,
            const Candidate< Poly > & rhs ) const  [inline], [inherited]
```

15.380 smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::LD > Struct Template Reference

```
#include <ProjectionComparator.h>
```

Public Member Functions

- bool operator() (const Candidate< Poly > &lhs, const Candidate< Poly > &rhs) const

15.380.1 Member Function Documentation

15.380.1.1 operator()()

```
bool smtrat::cad::projection_compare::ProjectionComparator_impl< Args >::operator() (
            const Candidate< Poly > & lhs,
            const Candidate< Poly > & rhs ) const  [inline], [inherited]
```

15.381 smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::PD > Struct Template Reference

```
#include <ProjectionComparator.h>
```

Public Member Functions

- bool operator() (const Candidate< Poly > &lhs, const Candidate< Poly > &rhs) const

15.381.1 Member Function Documentation

15.381.1.1 operator()()

```
bool smtrat::cad::projection_compare::ProjectionComparator_impl< Args >::operator() (
            const Candidate< Poly > & lhs,
            const Candidate< Poly > & rhs ) const  [inline], [inherited]
```

15.382 smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::SD > Struct Template Reference

```
#include <ProjectionComparator.h>
```

Public Member Functions

- bool operator() (const Candidate< Poly > &lhs, const Candidate< Poly > &rhs) const

15.382.1 Member Function Documentation

15.382.1.1 operator()()

```
bool smtrat::cad::projection_compare::ProjectionComparator_impl< Args >::operator() (
            const Candidate< Poly > & lhs,
            const Candidate< Poly > & rhs ) const  [inline], [inherited]
```

15.383 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

15.383.1 Member Function Documentation

15.383.1.1 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]
```

15.384 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

15.384.1 Member Typedef Documentation

15.384.1.1 ECMap

```
using smtrat::cad::ProjectionGlobalInformation::ECMap = std::map<Origin::BaseType,
ECData>
```

15.384.2 Member Function Documentation

15.384.2.1 addToEC()

```
void smtrat::cad::ProjectionGlobalInformation::addToEC (
            const Origin::BaseType & origin,
            std::size_t level,
            std::size_t pid )  [inline]
```

15.384.2.2 createEC()

```
void smtrat::cad::ProjectionGlobalInformation::createEC (
            const Origin::BaseType & origin )  [inline]
```

15.384.2.3 isEC()

```
bool smtrat::cad::ProjectionGlobalInformation::isEC (
    const Origin::BaseType & origin ) const [inline]
```

15.384.2.4 removeEC()

```
void smtrat::cad::ProjectionGlobalInformation::removeEC (
    const Origin::BaseType & origin ) [inline]
```

15.384.2.5 reset()

```
void smtrat::cad::ProjectionGlobalInformation::reset (
    std::size_t dim ) [inline]
```

15.384.3 Field Documentation

15.384.3.1 mECs

```
ECMap smtrat::cad::ProjectionGlobalInformation::mECs
```

15.384.3.2 mInactive

```
carl::Bitset smtrat::cad::ProjectionGlobalInformation::mInactive
```

15.384.3.3 mUsedEC

```
std::vector<ECMap::const_iterator> smtrat::cad::ProjectionGlobalInformation↩
::mUsedEC
```

15.385 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)

15.385.1 Member Function Documentation

15.385.1.1 addECConstraint()

```
void smtrat::cad::ProjectionInformation::addECConstraint (
    std::size_t pid ) [inline]
```

15.385.1.2 clear()

```
void smtrat::cad::ProjectionInformation::clear (
    std::size_t level,
    std::size_t pid ) [inline]
```

15.385.1.3 emplace()

```
void smtrat::cad::ProjectionInformation::emplace (
    std::size_t level,
    std::size_t pid ) [inline]
```

15.385.1.4 getUsedEC()

```
const carl::Bitset& smtrat::cad::ProjectionInformation::getUsedEC (
    std::size_t level ) const [inline]
```

15.385.1.5 hasEC()

```
bool smtrat::cad::ProjectionInformation::hasEC (
    std::size_t level ) const [inline]
```

15.385.1.6 hasInfo() [1/2]

```
bool smtrat::cad::ProjectionInformation::hasInfo (
    std::size_t level ) const [inline]
```

15.385.1.7 hasInfo() [2/2]

```
bool smtrat::cad::ProjectionInformation::hasInfo (
    std::size_t level,
    std::size_t pid ) const [inline]
```

15.385.1.8 isActive()

```
bool smtrat::cad::ProjectionInformation::isActive (
    std::size_t level,
    std::size_t id ) const [inline]
```

15.385.1.9 operator()() [1/6]

```
auto& smtrat::cad::ProjectionInformation::operator() ( ) [inline]
```

15.385.1.10 operator()() [2/6]

```
const auto& smtrat::cad::ProjectionInformation::operator() ( )
const [inline]
```

15.385.1.11 operator()() [3/6]

```
auto& smtrat::cad::ProjectionInformation::operator() (
    std::size_t level ) [inline]
```

15.385.1.12 operator()() [4/6]

```
const auto& smtrat::cad::ProjectionInformation::operator() (
    std::size_t level ) const [inline]
```

15.385.1.13 operator()() [5/6]

```
auto& smtrat::cad::ProjectionInformation::operator() (
    std::size_t level,
    std::size_t pid ) [inline]
```

15.385.1.14 operator()() [6/6]

```
const auto& smtrat::cad::ProjectionInformation::operator() (
    std::size_t level,
    std::size_t pid ) const [inline]
```

15.385.1.15 removeECConstraint()

```
void smtrat::cad::ProjectionInformation::removeECConstraint (
    std::size_t ) [inline]
```

15.385.1.16 reset()

```
void smtrat::cad::ProjectionInformation::reset (
    std::size_t dim ) [inline]
```

15.385.1.17 selectEC()

```
bool smtrat::cad::ProjectionInformation::selectEC (
    std::size_t level ) [inline]
```

15.385.1.18 unselectEC()

```
void smtrat::cad::ProjectionInformation::unselectEC (
    std::size_t level ) [inline]
```

15.385.1.19 usingEC()

```
bool smtrat::cad::ProjectionInformation::usingEC (
    std::size_t level ) const [inline]
```

15.386 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)

15.386.1 Member Function Documentation

15.386.1.1 emplace()

```
void smtrat::cad::ProjectionLevelInformation::emplace (
    std::size_t level ) [inline]
```

15.386.1.2 hasInfo()

```
bool smtrat::cad::ProjectionLevelInformation::hasInfo (
    std::size_t level ) const [inline]
```

15.386.1.3 operator()() [1/2]

```
auto& smtrat::cad::ProjectionLevelInformation::operator() (
    std::size_t level ) [inline]
```

15.386.1.4 operator()() [2/2]

```
const auto& smtrat::cad::ProjectionLevelInformation::operator() (
    std::size_t level ) const [inline]
```

15.386.1.5 reset()

```
void smtrat::cad::ProjectionLevelInformation::reset (
    std::size_t dim ) [inline]
```

15.387 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

15.387.1 Member Function Documentation

15.387.1.1 operator()() [1/2]

```
template<typename Callback >
void smtrat::cad::ProjectionOperator::operator() (
    ProjectionType pt,
    const UPoly & p,
    carl::Variable variable,
    Callback && cb ) const [inline]
```

15.387.1.2 operator()() [2/2]

```
template<typename Callback >
void smtrat::cad::ProjectionOperator::operator() (
    ProjectionType pt,
    const UPoly & p,
    const UPoly & q,
    carl::Variable variable,
    Callback && i ) const [inline]
```

15.388 smtrat::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)

15.388.1 Member Function Documentation

15.388.1.1 clear()

```
void smtrat::cad::ProjectionPolynomialInformation::clear (
    std::size_t level,
    std::size_t pid ) [inline]
```

15.388.1.2 emplace()

```
void smtrat::cad::ProjectionPolynomialInformation::emplace (
    std::size_t level,
    std::size_t pid ) [inline]
```

15.388.1.3 hasInfo()

```
bool smtrat::cad::ProjectionPolynomialInformation::hasInfo (
    std::size_t level,
    std::size_t pid ) const [inline]
```

15.388.1.4 operator()() [1/2]

```
auto& smtrat::cad::ProjectionPolynomialInformation::operator() (
    std::size_t level,
    std::size_t pid ) [inline]
```

15.388.1.5 operator()() [2/2]

```
const auto& smtrat::cad::ProjectionPolynomialInformation::operator()
(
    std::size_t level,
    std::size_t pid ) const [inline]
```

15.388.1.6 reset()

```
void smtrat::cad::ProjectionPolynomialInformation::reset (
    std::size_t ) [inline]
```

15.389 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 Poly &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)
- 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 Poly &, const Poly &)> compare) const
- std::size_t degree (PolyRef p)
- std::size_t max_degree (PolyRef p)

15.389.1 Detailed Description Encapsulates all computations on polynomials.

Computations are cached with respect to a PolyPool.

15.389.2 Constructor & Destructor Documentation

15.389.2.1 Projections() `smtrat::cadcells::datastructures::Projections::Projections ( PolyPool & pool ) [inline]`

15.389.3 Member Function Documentation

15.389.3.1 clear_assignment_cache() `void smtrat::cadcells::datastructures::Projections::clear_assignment_cache ( const Assignment & assignment ) [inline]`

Clears all projections cached with respect to this assignment.

15.389.3.2 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.

15.389.3.3 degree() `std::size_t smtrat::cadcells::datastructures::Projections::degree ( PolyRef p ) [inline]`

15.389.3.4 disc()

```
PolyRef smtrat::cadcells::datastructures::Projections::disc (
    PolyRef p ) [inline]
```

15.389.3.5 factors_nonconst()

```
const std::vector<PolyRef>& smtrat::cadcells::datastructures::↩
Projections::factors_nonconst (
    PolyRef p ) [inline]
```

15.389.3.6 has_const_coeff()

```
bool smtrat::cadcells::datastructures::Projections::has_const_coeff
(
    PolyRef p ) const [inline]
```

15.389.3.7 is_const()

```
bool smtrat::cadcells::datastructures::Projections::is_const (
    PolyRef p ) [inline]
```

15.389.3.8 is_disc_zero()

```
bool smtrat::cadcells::datastructures::Projections::is_disc_zero (
    const Assignment & sample,
    PolyRef p ) [inline]
```

15.389.3.9 is_ldcf_zero()

```
bool smtrat::cadcells::datastructures::Projections::is_ldcf_zero (
    const Assignment & sample,
    PolyRef p ) [inline]
```

15.389.3.10 is_nullified()

```
bool smtrat::cadcells::datastructures::Projections::is_nullified (
    const Assignment & sample,
    PolyRef p ) [inline]
```

15.389.3.11 is_zero() [1/2]

```
bool smtrat::cadcells::datastructures::Projections::is_zero (
    const Assignment & sample,
    PolyRef p ) [inline]
```

15.389.3.12 is_zero() [2/2]

```
bool smtrat::cadcells::datastructures::Projections::is_zero (
    PolyRef p ) [inline]
```

15.389.3.13 know_disc()

```
bool smtrat::cadcells::datastructures::Projections::know_disc (
    PolyRef p ) const [inline]
```

15.389.3.14 known()

```
bool smtrat::cadcells::datastructures::Projections::known (
    const Poly & p ) const  [inline]
```

15.389.3.15 ldcf()

```
PolyRef smtrat::cadcells::datastructures::Projections::ldcf (
    PolyRef p )  [inline]
```

15.389.3.16 main_var()

```
auto smtrat::cadcells::datastructures::Projections::main_var (
    PolyRef p ) const  [inline]
```

15.389.3.17 max_degree()

```
std::size_t smtrat::cadcells::datastructures::Projections::max_degree
(
    PolyRef p )  [inline]
```

15.389.3.18 num_roots()

```
size_t smtrat::cadcells::datastructures::Projections::num_roots (
    const Assignment & sample,
    PolyRef p )  [inline]
```

15.389.3.19 polys() [1/2]

```
auto& smtrat::cadcells::datastructures::Projections::polys ( )  [inline]
```

15.389.3.20 polys() [2/2]

```
const auto& smtrat::cadcells::datastructures::Projections::polys ( )
const  [inline]
```

15.389.3.21 real_roots()

```
std::vector<RAN> smtrat::cadcells::datastructures::Projections::real_↩
roots (
    const Assignment & sample,
    PolyRef p )  [inline]
```

15.389.3.22 res()

```
PolyRef smtrat::cadcells::datastructures::Projections::res (
    PolyRef p,
    PolyRef q )  [inline]
```

15.389.3.23 simplest_nonzero_coeff()

```
PolyRef smtrat::cadcells::datastructures::Projections↩
::simplest_nonzero_coeff (
    const Assignment & sample,
    PolyRef p,
    std::function< bool(const Poly &, const Poly &)> compare ) const  [inline]
```

15.390 smtrat::cadcells::operators::PropertiesSet< Op > Struct Template Reference

```
#include <operator.h>
```

15.391 smtrat::cadcells::operators::PropertiesSet< op::mccallum > Struct Template Reference

`#include <operator_mccallum.h>`

Public Types

- using type = datastructures::PropertiesT< properties::poly_sgn_inv, properties::poly_irreducible_sgn_inv, properties::poly_ord_inv, properties::root_well_def, properties::poly_pdel, properties::cell_connected >

15.391.1 Member Typedef Documentation

15.391.1.1 type

`using smtrat::cadcells::operators::PropertiesSet< op::mccallum >::type = datastructures::PropertiesT<properties::poly_sgn_inv,properties::poly_irreducible_sgn_inv,properties::poly_ord_i`

15.392 smtrat::cadcells::datastructures::PropertiesT< Ts > Struct Template Reference

Set of properties.

`#include <properties.h>`

15.392.1 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<<.

15.393 smtrat::cadcells::datastructures::PropertiesT< T, Ts... > Struct Template Reference

`#include <properties.h>`

Data Fields

- PropertiesTContent< T, T::is_flag >::type content

15.393.1 Field Documentation

15.393.1.1 content

`template<class T , class... Ts> PropertiesTContent<T, T::is_flag>::type smtrat::cadcells::datastructures::PropertiesT< T, Ts... >::content`

15.394 smtrat::cadcells::datastructures::PropertiesTContent< T, is_flag > Struct Template Reference

```
#include <properties.h>
```

15.395 smtrat::cadcells::datastructures::PropertiesTContent< T, false > Struct Template Reference

```
#include <properties.h>
```

Public Types

- using type = PropertiesTSet< T >

15.395.1 Member Typedef Documentation

15.395.1.1 type

```
template<typename T >
using smtrat::cadcells::datastructures::PropertiesTContent< T, false >::type = PropertiesTSet<T>
```

15.396 smtrat::cadcells::datastructures::PropertiesTContent< T, true > Struct Template Reference

```
#include <properties.h>
```

Public Types

- using type = bool

15.396.1 Member Typedef Documentation

15.396.1.1 type

```
template<typename T >
using smtrat::cadcells::datastructures::PropertiesTContent< T, true >::type = bool
```

15.397 smtrat::cadcells::datastructures::property_hash< T > Struct Template Reference

```
#include <properties.h>
```

Public Member Functions

- std::size_t operator() (const T &p) const

15.397.1 Member Function Documentation

15.397.1.1 operator()()

```
template<typename T >
std::size_t smtrat::cadcells::datastructures::property_hash< T >::operator() (
            const T & p ) const [inline]
```

15.398 smtrat::PseudoBoolEncoder Class Reference

Base class for a PseudoBoolean Encoder.

```
#include <PseudoBoolEncoder.h>
```

Public Member Functions

- boost::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 boost::optional< FormulaT > doEncode (const ConstraintT &constraint)=0
- FormulaT generateVarChain (const std::set< carl::Variable > &vars, carl::FormulaType type)

15.398.1 Detailed Description

Base class for a PseudoBoolean Encoder.

It takes a arithmetic constraint and converts it to a boolean Formula

15.398.2 Member Function Documentation

15.398.2.1 canEncode()

```
virtual bool smtrat::PseudoBoolEncoder::canEncode (
    const ConstraintT & constraint ) [pure virtual]
```

Implemented in smtrat::TotalizerEncoder, smtrat::ExactlyOneCommanderEncoder, smtrat::MixedSignEncoder, smtrat::RNSEncoder, smtrat::CardinalityEncoder, smtrat::LongFormulaEncoder, and smtrat::ShortFormulaEncoder.

15.398.2.2 doEncode()

```
virtual boost::optional<FormulaT> smtrat::PseudoBoolEncoder::doEncode (
    const ConstraintT & constraint ) [protected], [pure virtual]
```

Implemented in smtrat::TotalizerEncoder, smtrat::ExactlyOneCommanderEncoder, smtrat::MixedSignEncoder, smtrat::CardinalityEncoder, smtrat::ShortFormulaEncoder, smtrat::LongFormulaEncoder, and smtrat::RNSEncoder.

15.398.2.3 encode()

```
boost::optional< FormulaT > smtrat::PseudoBoolEncoder::encode (
    const ConstraintT & constraint )
```

Encodes an arbitrary constraint.

Returns encoded formula

15.398.2.4 encodingSize()

```
Rational smtrat::PseudoBoolEncoder::encodingSize (
    const ConstraintT & constraint ) [virtual]
```

Reimplemented in smtrat::TotalizerEncoder, smtrat::ExactlyOneCommanderEncoder, smtrat::MixedSignEncoder, smtrat::ShortFormulaEncoder, smtrat::LongFormulaEncoder, and smtrat::CardinalityEncoder.

15.398.2.5 generateVarChain()

```
FormulaT smtrat::PseudoBoolEncoder::generateVarChain (
    const std::set< carl::Variable > & vars,
    carl::FormulaType type ) [protected]
```

15.398.2.6 name()

```
virtual std::string smtrat::PseudoBoolEncoder::name ( ) [inline], [virtual]
```

Reimplemented in smtrat::TotalizerEncoder, smtrat::ExactlyOneCommanderEncoder, smtrat::MixedSignEncoder, smtrat::CardinalityEncoder, smtrat::ShortFormulaEncoder, and smtrat::LongFormulaEncoder.

15.398.3 Field Documentation

15.398.3.1 problem_size `std::size_t smtrat::PseudoBoolEncoder::problem_size`

15.399 smtrat::PseudoBoolNormalizer Class Reference

`#include <PseudoBoolNormalizer.h>`

Public Member Functions

- std::pair< boost::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)

15.399.1 Member Function Documentation

15.399.1.1 normalize()

```
std::pair< boost::optional< FormulaT >, ConstraintT > smtrat::PseudoBoolNormalizer::normalize (
            const ConstraintT & constraint )
```

Modifies the constraint in-place and returns an additional boolean formula.

Consider -4x <= 2

This is equivalent to -4∗(1 - not x) <= 2 iff. -4 + 4 not x <= 2 iff 4 not x <= 6 Since we can not represent this negation in the constraint itself we add a variable y and get y <-> not x and 4 y <= 6

In this particular case we can later on remove y again since the constraint is trivially satisfied.

15.399.1.2 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 occurences of the substituted variable should be substituted as well or in general when the correlation is needed.

15.399.1.3 trim()

```
ConstraintT smtrat::PseudoBoolNormalizer::trim (
            const ConstraintT & constraint )
```

15.400 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

15.400.1 Constructor & Destructor Documentation

15.400.1.1 QEParser()

```
smtrat::parser::QEParser::QEParser (
    Theories * theories ) [inline]
```

15.400.2 Member Function Documentation

15.400.2.1 resolveVariable()

```
carl::Variable smtrat::parser::QEParser::resolveVariable (
    const Identifier & name ) const [inline]
```

15.400.3 Field Documentation

15.400.3.1 main

```
qi::rule<Iterator, qe::QEQuery(), Skipper> smtrat::parser::QEParser::main
```

15.400.3.2 qualifiedidentifier

```
QualifiedIdentifierParser smtrat::parser::QEParser::qualifiedidentifier
```

15.400.3.3 quantifier

```
QuantifierParser smtrat::parser::QEParser::quantifier
```

15.400.3.4 theories

```
Theories* smtrat::parser::QEParser::theories
```

15.400.3.5 var

```
qi::rule<Iterator, carl::Variable(), Skipper> smtrat::parser::QEParser::var
```

15.401 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

15.401.1 Constructor & Destructor Documentation

15.401.1.1 QualifiedIdentifierParser()

```
smtrat::parser::QualifiedIdentifierParser::Qualified↩
IdentifierParser ( ) [inline]
```

15.401.2 Member Function Documentation

15.401.2.1 checkQualification()

```
Identifier smtrat::parser::QualifiedIdentifierParser::check↩
Qualification (
    const Identifier & identifier,
    const carl::Sort & ) const [inline]
```

Todo Check what can be checked here.

15.401.3 Field Documentation

15.401.3.1 identifier `IdentifierParser smtrat::parser::QualifiedIdentifierParser::identifier`

15.401.3.2 main `qi::rule<Iterator, Identifier(), Skipper> smtrat::parser::QualifiedIdentifierParser::main`

15.401.3.3 sort `SortParser smtrat::parser::QualifiedIdentifierParser::sort`

15.402 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

15.402.1 Constructor & Destructor Documentation

15.402.1.1 QuantifierExpression()

```
smtrat::expression::QuantifierExpression::QuantifierExpression
(
    QuantifierType _type,
    std::vector< carl::Variable > && _variables,
    Expression && _expression )  [inline]
```

15.402.2 Field Documentation

15.402.2.1 expression `Expression smtrat::expression::QuantifierExpression::expression`

15.402.2.2 type `QuantifierType smtrat::expression::QuantifierExpression::type`

15.402.2.3 variables `std::vector<carl::Variable> smtrat::expression::QuantifierExpression::variables`

15.403 smtrat::parser::QuantifierParser Struct Reference

`#include <Script.h>`

Public Member Functions

- QuantifierParser ()

15.403.1 Constructor & Destructor Documentation

15.403.1.1 QuantifierParser()

```
smtrat::parser::QuantifierParser::QuantifierParser ( )  [inline]
```

15.404 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)

15.404.1 Member Typedef Documentation

15.404.1.1 Key

```
template<class T >
typedef T Minisat::Queue< T >::Key
```

15.404.2 Constructor & Destructor Documentation

15.404.2.1 Queue()

```
template<class T >
Minisat::Queue< T >::Queue ( ) [inline]
```

15.404.3 Member Function Documentation

15.404.3.1 clear()

```
template<class T >
void Minisat::Queue< T >::clear (
            bool dealloc = false ) [inline]
```

15.404.3.2 insert()

```
template<class T >
void Minisat::Queue< T >::insert (
            T elem ) [inline]
```

15.404.3.3 operator[]() [1/2]

```
template<class T >
T& Minisat::Queue< T >::operator[] (
            int index ) [inline]
```

15.404.3.4 operator[]() [2/2]

```
template<class T >
const T& Minisat::Queue< T >::operator[] (
            int index ) const [inline]
```

15.404.3.5 peek()

```
template<class T >
T Minisat::Queue< T >::peek ( ) const [inline]
```

15.404.3.6 pop() `template<class T >`

`void Minisat::Queue< T >::pop ( ) [inline]`

15.404.3.7 size() `template<class T >`

`int Minisat::Queue< T >::size ( ) const [inline]`

15.405 smtrat::mcsat::icp::QueueEntry Struct Reference

`#include <IntervalPropagation.h>`

Data Fields

- double priority
- carl::contractor::Contractor< FormulaT, Poly > contractor

15.405.1 Field Documentation

15.405.1.1 contractor `carl::contractor::Contractor<FormulaT, Poly> smtrat::mcsat::icp::Queue↩ Entry::contractor`

15.405.1.2 priority `double smtrat::mcsat::icp::QueueEntry::priority`

15.406 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.

15.406.1 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.

15.406.2 Constructor & Destructor Documentation

15.406.2.1 RandomizationAdaptor()

```
template<typename T >
benchmax::RandomizationAdaptor< T >::RandomizationAdaptor (
            const std::vector< T > & data ) [inline]
```

Construct a randomized view on the given vector.

15.406.3 Member Function Documentation

15.406.3.1 begin()

```
template<typename T >
auto benchmax::RandomizationAdaptor< T >::begin ( ) const [inline]
```

Return begin of the randomized range.

15.406.3.2 end()

```
template<typename T >
auto benchmax::RandomizationAdaptor< T >::end ( ) const [inline]
```

Return end of the randomized range.

15.407 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

15.407.1 Constructor & Destructor Documentation

15.407.1.1 RationalCapsule()

```
smtrat::RationalCapsule::RationalCapsule (
            const Rational & aRational,
            const Rational & bRational,
            const Rational & cRational )
```

15.407.2 Member Function Documentation

15.407.2.1 getARational()

```
const smtrat::Rational & smtrat::RationalCapsule::getARational ( )
const
```

15.407.2.2 getBRational()

```
const smtrat::Rational & smtrat::RationalCapsule::getBRational ( )
const
```

15.407.2.3 getCRational()

```
const smtrat::Rational & smtrat::RationalCapsule::getCRational ( )
const
```

15.407.3 Field Documentation

15.407.3.1 aRational `Rational smtrat::RationalCapsule::aRational = aRational`

15.407.3.2 bRational `Rational smtrat::RationalCapsule::bRational = bRational`

15.407.3.3 cRational `Rational smtrat::RationalCapsule::cRational = cRational`

15.408 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 &)

15.408.1 Detailed Description

Specialization of qi::real_policies for a Rational.

Specifies that neither NaN nor Inf is allowed.

15.408.2 Member Function Documentation

15.408.2.1 parse_inf()

```
template<typename It , typename Attr >
static bool smtrat::parser::RationalPolicies::parse_inf (
            It & ,
            It const & ,
            Attr & ) [inline], [static]
```

15.408.2.2 parse_nan()

```
template<typename It , typename Attr >
static bool smtrat::parser::RationalPolicies::parse_nan (
            It & ,
            It const & ,
            Attr & ) [inline], [static]
```

15.409 smtrat::fixedsize_allocator< T >::rebind< U > Struct Template Reference

`#include <alloc.h>`

Public Types

- typedef fixedsize_allocator< U > other

15.409.1 Member Typedef Documentation

15.409.1.1 other

```
template<typename T >
template<class U >
typedef fixedsize_allocator<U> smtrat::fixedsize_allocator< T >::rebind< U >::other
```

15.410 smtrat::mcsat::onecellcad::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 carl::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 carl::RealAlgebraicPoint< Rational > & point

15.410.1 Member Function Documentation

15.410.1.1 coeffNonNull()

```
std::optional<Poly> smtrat::mcsat::onecellcad::OneCellCAD::coeff↩
NonNull (
            const TagPoly & boundCandidate ) [inline], [inherited]
```

15.410.1.2 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.

15.410.1.3 isAlreadyProcessed()

```
bool smtrat::mcsat::onecellcad::recursive::RecursiveCAD::isAlreadyProcessed (
            const TagPoly & poly ) [inline]
```

15.410.1.4 isMainPointInsideCell()

```
bool smtrat::mcsat::onecellcad::OneCellCAD::isMainPointInsideCell (
            const CADCell & cell ) [inline], [inherited]
```

15.410.1.5 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(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!).

15.410.1.6 isPointRootOfPoly() [1/2]

```
bool smtrat::mcsat::onecellcad::OneCellCAD::isPointRootOfPoly (
            const std::size_t polyLevel,
            const Poly & poly ) [inline], [inherited]
```

15.410.1.7 isPointRootOfPoly() [2/2]

```
bool smtrat::mcsat::onecellcad::OneCellCAD::isPointRootOfPoly (
            const TagPoly & poly ) [inline], [inherited]
```

15.410.1.8 OneCellCAD()

```
smtrat::mcsat::onecellcad::OneCellCAD::OneCellCAD [inline]
```

15.410.1.9 partialDerivativesLayerWithSomeNonEarlyVanishingPoly()

```
std::vector<Poly> smtrat::mcsat::onecellcad::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

poly	must not be a const-poly like 2, otherwise this function does not terminate.

Returns Actually only returns the set S

15.410.1.10 pointEnclosingCADCell()

```
std::optional<CADCell> smtrat::mcsat::onecellcad::recursive↩
::RecursiveCAD::pointEnclosingCADCell (
            const std::vector< std::vector< onecellcad::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

variableOrder	must contain unique variables and at least one, because constant polynomials (without a variable) are prohibited.
point	point.size() >= variables.size().
polys	must contain only non-constant, irreducible polynomials that mention only variables that appear in 'variableOrder'.

15.410.1.11 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'.

15.410.1.12 refineNonNull()

```
ShrinkResult smtrat::mcsat::onecellcad::recursive::RecursiveCAD↩
::refineNonNull (
            const TagPoly & boundCandidate,
            CADCell & cell ) [inline]
```

15.410.1.13 refineNull()

```
ShrinkResult smtrat::mcsat::onecellcad::recursive::RecursiveCAD::refine↩
Null (
            const TagPoly & boundCandidate,
            CADCell & cell ) [inline]
```

15.410.1.14 shrinkCell()

```
ShrinkResult smtrat::mcsat::onecellcad::recursive::RecursiveCAD::shrink↩
Cell (
            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)

15.410.1.15 shrinkCellWithEarlyVanishingPoly()

```
ShrinkResult smtrat::mcsat::onecellcad::recursive↩
::RecursiveCAD::shrinkCellWithEarlyVanishingPoly (
            const TagPoly & boundCandidate,
            CADCell & cell ) [inline]
```

15.410.1.16 shrinkCellWithIrreducibleFactorsOfPoly()

```
ShrinkResult smtrat::mcsat::onecellcad↩
::recursive::RecursiveCAD::shrinkCellWithIrreducibleFactorsOfPoly (
            const TagPoly & poly,
            CADCell & cell ) [inline]
```

15.410.1.17 shrinkCellWithNonRootPoint()

```
ShrinkResult smtrat::mcsat::onecellcad::recursive::↩
RecursiveCAD::shrinkCellWithNonRootPoint (
            const TagPoly & boundCandidate,
            CADCell & cell ) [inline]
```

15.410.1.18 shrinkCellWithPolyHavingPointAsRoot()

```
ShrinkResult smtrat::mcsat::onecellcad↩
::recursive::RecursiveCAD::shrinkCellWithPolyHavingPointAsRoot (
            const TagPoly & boundCandidate,
            CADCell & cell ) [inline]
```

15.410.1.19 shrinkSingleComponent()

```
void smtrat::mcsat::onecellcad::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

boundCandidate	Must be a non-const irreducible polynomial that does not vanish early.

15.410.1.20 vanishesEarly()

```
bool smtrat::mcsat::onecellcad::OneCellCAD::vanishesEarly (
    const std::size_t polyLevel,
    const Poly & poly ) [inline], [inherited]
```

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'.

15.410.2 Field Documentation

15.410.2.1 point

```
const carl::RealAlgebraicPoint<Rational>& smtrat::mcsat::onecellcad::OneCellCAD::point [inherited]
```

15.410.2.2 variableOrder

```
const std::vector<carl::Variable>& smtrat::mcsat::onecellcad::OneCellCAD::variableOrder [inherited]
```

Variables can also be indexed by level.

Polys with mathematical level 1 contain the variable in variableOrder[0]

15.411 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.

15.411.1 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::

15.411.2 Constructor & Destructor Documentation

15.411.2.1 Reducta()

```
template<typename Poly>
smtrat::cad::projection::Reducta< Poly >::Reducta (
    const Poly & p ) [inline]
```

15.411.3 Field Documentation

15.411.3.1 elements `T std::vector< T >::elements [inherited]`

STL member.

15.412 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)

15.412.1 Member Typedef Documentation

15.412.1.1 Ref `template<class T>`

`typedef uint32_t Minisat::RegionAllocator< T >::Ref`

15.412.2 Member Enumeration Documentation

15.412.2.1 anonymous enum `template<class T>`

`anonymous enum`

Enumerator

Ref_Undef	

15.412.2.2 anonymous enum `template<class T>`

`anonymous enum`

Enumerator

Unit_Size	

15.412.3 Constructor & Destructor Documentation

15.412.3.1 RegionAllocator() `template<class T>`

```
Minisat::RegionAllocator< T >::RegionAllocator (
            uint32_t start_cap = 1024 * 1024 ) [inline], [explicit]
```

15.412.3.2 ~RegionAllocator() template<class T>

```
Minisat::RegionAllocator< T >::~RegionAllocator ( ) [inline]
```

15.412.4 Member Function Documentation

15.412.4.1 ael() template<class T>

```
Ref Minisat::RegionAllocator< T >::ael (
            const T * t ) [inline]
```

15.412.4.2 alloc() template<class T >

```
RegionAllocator< T >::Ref Minisat::RegionAllocator< T >::alloc (
            int size )
```

15.412.4.3 free() template<class T>

```
void Minisat::RegionAllocator< T >::free (
            int size ) [inline]
```

15.412.4.4 lea() [1/2] template<class T>

```
T* Minisat::RegionAllocator< T >::lea (
            Ref r ) [inline]
```

15.412.4.5 lea() [2/2] template<class T>

```
const T* Minisat::RegionAllocator< T >::lea (
            Ref r ) const [inline]
```

15.412.4.6 moveTo() template<class T>

```
void Minisat::RegionAllocator< T >::moveTo (
            RegionAllocator< T > & to ) [inline]
```

15.412.4.7 operator[]() [1/2] template<class T>

```
T& Minisat::RegionAllocator< T >::operator[] (
            Ref r ) [inline]
```

15.412.4.8 operator[]() [2/2] template<class T>

```
const T& Minisat::RegionAllocator< T >::operator[] (
            Ref r ) const [inline]
```

15.412.4.9 size() template<class T>

```
uint32_t Minisat::RegionAllocator< T >::size ( ) const [inline]
```

15.412.4.10 wasted() `template<class T>`

`uint32_t Minisat::RegionAllocator< T >::wasted ( ) const [inline]`

15.413 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 ()

15.413.1 Constructor & Destructor Documentation

15.413.1.1 remove_xor_first_arg()

```
smtrat::NNFRewriter::remove_xor_first_arg::remove_xor_first_arg (
    NNFRewriter & r_,
    const FormulaT & f_ ) [inline]
```

15.413.1.2 ~remove_xor_first_arg()

`smtrat::NNFRewriter::remove_xor_first_arg::~remove_xor_first_arg ( ) [inline]`

15.414 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)

15.414.1 Member Typedef Documentation

15.414.1.1 term_type

`typedef carl::UTerm smtrat::ReplaceVariablesRewriter::term_type`

15.414.1.2 UEquality

`typedef carl::UEquality smtrat::ReplaceVariablesRewriter::UEquality`

15.414.1.3 UFInstance

`typedef carl::UFInstance smtrat::ReplaceVariablesRewriter::UFInstance`

15.414.1.4 UninterpretedFunction

`typedef carl::UninterpretedFunction smtrat::ReplaceVariablesRewriter::Uninter`

15.414.1.5 UVariable `typedef carl::UVariable smtrat::ReplaceVariablesRewriter::UVariable`

15.414.2 Constructor & Destructor Documentation

15.414.2.1 ReplaceVariablesRewriter()

```
smtrat::ReplaceVariablesRewriter::ReplaceVariablesRewriter (
    std::unordered_map< FormulaT, bool > & facts,
    ModuleWrapper< EQModule< EQSettingsForPreprocessing >> & helper ) [inline]
```

15.414.3 Member Function Documentation

15.414.3.1 enter_and()

```
void smtrat::ReplaceVariablesRewriter::enter_and (
    const FormulaT & ) [inline]
```

15.414.3.2 enter_default()

```
void smtrat::ReplaceVariablesRewriter::enter_default (
    const FormulaT & ) [inline]
```

15.414.3.3 rewrite_and()

```
FormulaT smtrat::ReplaceVariablesRewriter::rewrite_and (
    const FormulaT & ,
    FormulaSetT && subformulas ) [inline]
```

15.414.3.4 rewrite_default()

```
FormulaT smtrat::ReplaceVariablesRewriter::rewrite_default (
    const FormulaT & ,
    const FormulaT & result ) [inline]
```

15.414.3.5 rewrite_ueq()

```
FormulaT smtrat::ReplaceVariablesRewriter::rewrite_ueq (
    const FormulaT & ueq ) [inline]
```

15.415 delta::ErrorHandler::result< typename > Struct Template Reference

`#include <Parser.h>`

Public Types

- typedef qi::error_handler_result type

15.415.1 Member Typedef Documentation

15.415.1.1 type

```
template<typename >
typedef qi::error_handler_result delta::ErrorHandler::result< typename >::type
```

15.416 smtrat::parser::ErrorHandler::result< typename > Struct Template Reference

`#include <Script.h>`

Public Types

- typedef qi::error_handler_result type

15.416.1 Member Typedef Documentation

15.416.1.1 type `template<typename >`

`typedef qi::error_handler_result smtrat::parser::ErrorHandler::result< typename >::type`

15.417 smtrat::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

15.417.1 Constructor & Destructor Documentation

15.417.1.1 ResultantRule()

```
smtrat::cad::preprocessor::ResultantRule::ResultantRule (
    Origins & origins,
    const std::vector< carl::Variable > & vars )  [inline]
```

15.417.2 Member Function Documentation

15.417.2.1 compute()

```
std::optional< std::vector< FormulaT > > smtrat::cad::preprocessor::↩
ResultantRule::compute (
    const std::map< ConstraintT, ConstraintT > & constraints )
```

15.417.2.2 getNewECs()

`const auto& smtrat::cad::preprocessor::ResultantRule::getNewECs ( ) const [inline]`

15.418 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.

15.418.1 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.

15.418.2 Member Function Documentation

15.418.2.1 addResult()

```
void benchmax::Results::addResult (
    const Tool * tool,
    const fs::path & file,
    BenchmarkResult && results ) [inline]
```

Add a new result.

15.418.2.2 data()

```
const auto& benchmax::Results::data ( ) const [inline]
```

15.418.2.3 get()

```
std::optional<std::reference_wrapper<const BenchmarkResult> > benchmax::↩
Results::get (
    const Tool * tool,
    const fs::path & file ) const [inline]
```

15.418.2.4 store() [1/2]

```
void benchmax::Results::store (
    Database & db ) [inline]
```

Store all results to some database.

15.418.2.5 store() [2/2]

```
void benchmax::Results::store (
    XMLWriter & xml,
    const Jobs & jobs ) const [inline]
```

Store all results to a xml file.

15.419 smtrat::RNSEncoder Class Reference

```
#include <RNSEncoder.h>
```

Public Member Functions

- RNSEncoder ()
- bool canEncode (const ConstraintT &constraint)
- boost::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

- boost::optional< FormulaT > doEncode (const ConstraintT &constraint)
- FormulaT generateVarChain (const std::set< carl::Variable > &vars, carl::FormulaType type)

15.419.1 Constructor & Destructor Documentation

15.419.1.1 RNSEncoder()

```
smtrat::RNSEncoder::RNSEncoder ( ) [inline]
```

15.419.2 Member Function Documentation

15.419.2.1 canEncode()

```
bool smtrat::RNSEncoder::canEncode (
    const ConstraintT & constraint ) [virtual]
```

Implements smtrat::PseudoBoolEncoder.

15.419.2.2 doEncode()

```
boost::optional< FormulaT > smtrat::RNSEncoder::doEncode (
    const ConstraintT & constraint ) [protected], [virtual]
```

Implements smtrat::PseudoBoolEncoder.

15.419.2.3 encode()

```
boost::optional< FormulaT > smtrat::PseudoBoolEncoder::encode (
    const ConstraintT & constraint ) [inherited]
```

Encodes an arbitrary constraint.

Returns encoded formula

15.419.2.4 encodingSize()

```
Rational smtrat::PseudoBoolEncoder::encodingSize (
    const ConstraintT & constraint ) [virtual], [inherited]
```

Reimplemented in smtrat::TotalizerEncoder, smtrat::ExactlyOneCommanderEncoder, smtrat::MixedSignEncoder, smtrat::ShortFormulaEncoder, smtrat::LongFormulaEncoder, and smtrat::CardinalityEncoder.

15.419.2.5 generateVarChain()

```
FormulaT smtrat::PseudoBoolEncoder::generateVarChain (
    const std::set< carl::Variable > & vars,
    carl::FormulaType type ) [protected], [inherited]
```

15.419.2.6 name()

```
virtual std::string smtrat::PseudoBoolEncoder::name ( ) [inline], [virtual],
[inherited]
```

Reimplemented in smtrat::TotalizerEncoder, smtrat::ExactlyOneCommanderEncoder, smtrat::MixedSignEncoder, smtrat::CardinalityEncoder, smtrat::ShortFormulaEncoder, and smtrat::LongFormulaEncoder.

15.419.3 Field Documentation

15.419.3.1 problem_size

```
std::size_t smtrat::PseudoBoolEncoder::problem_size [inherited]
```

15.420 smtrat::mcsat::onecellcad::firstlevel::root_indexer Class Reference

```
#include <first_level.h>
```

Public Member Functions

- void insert_root (const RAN &ran, const indexed_root &&root)
- void process ()
- const auto & samples ()
- const auto & roots ()
- const auto & indexed_roots_at (const RAN &ran)

Friends

- std::ostream & operator<< (std::ostream &os, const root_indexer::sample &data)
- std::ostream & operator<< (std::ostream &os, const root_indexer &data)

15.420.1 Member Function Documentation

15.420.1.1 indexed_roots_at()

```
const auto& smtrat::mcsat::onecellcad::firstlevel::root_indexer↩
::indexed_roots_at (
            const RAN & ran ) [inline]
```

15.420.1.2 insert_root()

```
void smtrat::mcsat::onecellcad::firstlevel::root_indexer::insert_root (
            const RAN & ran,
            const indexed_root && root ) [inline]
```

15.420.1.3 process()

```
void smtrat::mcsat::onecellcad::firstlevel::root_indexer::process ( )
[inline]
```

15.420.1.4 roots()

```
const auto& smtrat::mcsat::onecellcad::firstlevel::root_indexer::roots ( )
[inline]
```

15.420.1.5 samples()

```
const auto& smtrat::mcsat::onecellcad::firstlevel::root_indexer::samples (
) [inline]
```

15.420.2 Friends And Related Function Documentation

15.420.2.1 operator<< [1/2]

```
std::ostream& operator<< (
            std::ostream & os,
            const root_indexer & data ) [friend]
```

15.420.2.2 operator<< [2/2]

```
std::ostream& operator<< (
            std::ostream & os,
            const root_indexer::sample & data ) [friend]
```

15.421 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

15.421.1 Member Function Documentation

15.421.1.1 hash_on_level() `std::size_t smtrat::cadcells::operators::properties::root_well_def↩ ::hash_on_level ( ) const [inline]`

15.421.1.2 level() `size_t smtrat::cadcells::operators::properties::root_well_def::level ( ) const [inline]`

15.421.2 Field Documentation

15.421.2.1 is_flag `constexpr bool smtrat::cadcells::operators::properties::root_well_def::is_flag = false [static], [constexpr]`

15.421.2.2 root `datastructures::IndexedRoot smtrat::cadcells::operators::properties::root_↩ well_def::root`

15.422 smtrat::mcsat::arithmetic::RootIndexer Class Reference

`#include <RootIndexer.h>`

Public Types

- using RAN = carl::RealAlgebraicNumber< Rational >

Public Member Functions

- void add (const std::vector< RAN > &list)
- void process ()
- std::size_t size () const
- std::size_t operator[ ] (const RAN &ran) const
- const RAN & operator[ ] (std::size_t n) const
- const RAN & sampleFrom (std::size_t n) const
- bool is_root (std::size_t n) const

15.422.1 Member Typedef Documentation

15.422.1.1 RAN `using smtrat::mcsat::arithmetic::RootIndexer::RAN = carl::RealAlgebraic↩ Number<Rational>`

15.422.2 Member Function Documentation

15.422.2.1 add() `void smtrat::mcsat::arithmetic::RootIndexer::add ( const std::vector< RAN > & list ) [inline]`

15.422.2.2 is_root() `bool smtrat::mcsat::arithmetic::RootIndexer::is_root ( std::size_t n ) const [inline]`

15.422.2.3 operator[]() [1/2] `std::size_t smtrat::mcsat::arithmetic::RootIndexer::operator[] ( const RAN & ran ) const [inline]`

15.422.2.4 operator[]() [2/2]

```
const RAN& smtrat::mcsat::arithmetic::RootIndexer::operator[] (
    std::size_t n ) const [inline]
```

15.422.2.5 process()

```
void smtrat::mcsat::arithmetic::RootIndexer::process ( ) [inline]
```

15.422.2.6 sampleFrom()

```
const RAN& smtrat::mcsat::arithmetic::RootIndexer::sampleFrom (
    std::size_t n ) const [inline]
```

15.422.2.7 size()

```
std::size_t smtrat::mcsat::arithmetic::RootIndexer::size ( ) const [inline]
```

15.423 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)

15.423.1 Constructor & Destructor Documentation

15.423.1.1 Sample() [1/3]

```
smtrat::cad::Sample::Sample (
    const RAN & value ) [inline], [explicit]
```

15.423.1.2 Sample() [2/3]

```
smtrat::cad::Sample::Sample (
    const RAN & value,
    bool isRoot ) [inline], [explicit]
```

15.423.1.3 Sample() [3/3]

```
smtrat::cad::Sample::Sample (
    const RAN & value,
    std::size_t id ) [inline], [explicit]
```

15.423.2 Member Function Documentation

15.423.2.1 evaluatedWith() [1/2]

```
carl::Bitset& smtrat::cad::Sample::evaluatedWith ( ) [inline]
```

15.423.2.2 evaluatedWith() [2/2]

```
const carl::Bitset& smtrat::cad::Sample::evaluatedWith ( )
const [inline]
```

15.423.2.3 evaluationResult() [1/2]

```
carl::Bitset& smtrat::cad::Sample::evaluationResult ( )
[inline]
```

15.423.2.4 evaluationResult() [2/2]

```
const carl::Bitset& smtrat::cad::Sample::evaluationResult (
) const [inline]
```

15.423.2.5 getConflictingConstraints()

```
auto smtrat::cad::Sample::getConflictingConstraints ( )
const [inline]
```

15.423.2.6 hasConflictWithConstraint()

```
bool smtrat::cad::Sample::hasConflictWithConstraint ( )
const [inline]
```

15.423.2.7 isRoot()

```
bool smtrat::cad::Sample::isRoot ( ) const [inline]
```

15.423.2.8 liftedWith() [1/2]

```
SampleLiftedWith& smtrat::cad::Sample::liftedWith ( ) [inline]
```

15.423.2.9 liftedWith() [2/2]

```
const SampleLiftedWith& smtrat::cad::Sample::liftedWith ( ) const
[inline]
```

15.423.2.10 merge()

```
void smtrat::cad::Sample::merge (
    const Sample & s ) [inline]
```

15.423.2.11 operator<()

```
bool smtrat::cad::Sample::operator< (
    const Sample & s ) const [inline]
```

15.423.2.12 operator==()

```
bool smtrat::cad::Sample::operator== (
    const Sample & s ) const [inline]
```

15.423.2.13 operator>()

```
bool smtrat::cad::Sample::operator> (
    const Sample & s ) const [inline]
```

15.423.2.14 rootOf() [1/2]

```
SampleRootOf& smtrat::cad::Sample::rootOf ( ) [inline]
```

15.423.2.15 rootOf() [2/2]

```
const SampleRootOf& smtrat::cad::Sample::rootOf ( ) const [inline]
```

15.423.2.16 setIsRoot()

```
void smtrat::cad::Sample::setIsRoot (
    bool isRoot ) [inline]
```

15.423.2.17 value()

```
const RAN& smtrat::cad::Sample::value ( ) const [inline]
```

15.423.3 Friends And Related Function Documentation

15.423.3.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const Sample & s ) [friend]
```

15.424 smtrat::cad::sample_compare::SampleComparator< Iterator, Strategy > Struct Template Reference

```
#include <SampleComparator.h>
```

15.425 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

15.425.1 Member Function Documentation

15.425.1.1 operator()()

```
bool smtrat::cad::sample_compare::SampleComparator_impl< Iterator , Args
>::operator() (
    const Iterator & lhs,
    const Iterator & rhs ) const [inline], [inherited]
```

15.426 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

15.426.1 Member Function Documentation

15.426.1.1 operator()()

```
bool smtrat::cad::sample_compare::SampleComparator_impl< Iterator , Args >::operator() (
            const Iterator & lhs,
            const Iterator & rhs ) const  [inline], [inherited]
```

15.427 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

15.427.1 Member Function Documentation

15.427.1.1 operator()()

```
bool smtrat::cad::sample_compare::SampleComparator_impl< Iterator , Args >::operator() (
            const Iterator & lhs,
            const Iterator & rhs ) const  [inline], [inherited]
```

15.428 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

15.428.1 Member Function Documentation

15.428.1.1 operator()()

```
bool smtrat::cad::sample_compare::SampleComparator_impl< Iterator , Args >::operator() (
            const Iterator & lhs,
            const Iterator & rhs ) const  [inline], [inherited]
```

15.429 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

15.429.1 Member Function Documentation

15.429.1.1 operator()()

```
bool smtrat::cad::sample_compare::SampleComparator_impl< Iterator , Args >::operator() (
            const Iterator & lhs,
            const Iterator & rhs ) const  [inline], [inherited]
```

15.430 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

15.430.1 Member Function Documentation

15.430.1.1 operator()()

```
bool smtrat::cad::sample_compare::SampleComparator_impl< Iterator , Args >::operator() (
            const Iterator & lhs,
            const Iterator & rhs ) const  [inline], [inherited]
```

15.431 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

15.431.1 Member Function Documentation

15.431.1.1 operator()()

```
bool smtrat::cad::sample_compare::SampleComparator_impl< Iterator , Args >::operator() (
            const Iterator & lhs,
            const Iterator & rhs ) const  [inline], [inherited]
```

15.432 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

15.432.1 Member Function Documentation

15.432.1.1 operator()()

```
bool smtrat::cad::sample_compare::SampleComparator_impl< Iterator , Args >::operator() (
            const Iterator & lhs,
            const Iterator & rhs ) const  [inline], [inherited]
```

15.433 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

15.433.1 Member Function Documentation

15.433.1.1 operator()()

```
bool smtrat::cad::sample_compare::SampleComparator_impl< Iterator , Args >::operator() (
            const Iterator & lhs,
            const Iterator & rhs ) const  [inline], [inherited]
```

15.434 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

15.434.1 Member Function Documentation

15.434.1.1 operator()()

```
bool smtrat::cad::sample_compare::SampleComparator_impl< Iterator , Args >::operator() (
            const Iterator & lhs,
            const Iterator & rhs ) const  [inline], [inherited]
```

15.435 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

15.435.1 Member Function Documentation

15.435.1.1 compare()

```
template<typename Iterator >
bool smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Type >↩
::compare (
            const Iterator & lhs,
            const Iterator & rhs ) const  [inline]
```

15.435.1.2 operator()()

```
template<typename Iterator >
bool smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Type >↩
::operator() (
            const Iterator & lhs,
            const Iterator & rhs ) const  [inline]
```

15.435.1.3 reference()

```
template<typename Iterator >
bool smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Type >↩
::reference (
            const Iterator & lhs,
            const Iterator & rhs ) const  [inline]
```

15.436 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

15.436.1 Member Function Documentation

15.436.1.1 operator()()

```
bool smtrat::cad::sample_compare::SampleComparator_impl< Iterator , Args >::operator() (
            const Iterator & lhs,
            const Iterator & rhs ) const  [inline], [inherited]
```

15.437 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

15.437.1 Member Function Documentation

15.437.1.1 operator()()

```
template<typename It, typename... Args>
bool smtrat::cad::sample_compare::SampleComparator_impl< It, Args >::operator() (
            const It & lhs,
            const It & rhs ) const  [inline]
```

15.438 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 ()
- 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 ()
- 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)

15.438.1 Detailed Description

template<typename Properties>
class smtrat::cadcells::datastructures::SampledDerivation< Properties >

A SampledDerivation is a DelineatedDerivation with a sample and an DelineationInterval w.r.t.

to this sample.

Template Parameters

Properties	Set of properties

15.438.2 Constructor & Destructor Documentation

15.438.2.1 SampledDerivation()

```
template<typename Properties>
smtrat::cadcells::datastructures::SampledDerivation< Properties >::SampledDerivation (
            DelineatedDerivationRef< Properties > base,
            RAN main_sample )  [inline]
```

15.438.3 Member Function Documentation

15.438.3.1 base() [1/2]

```
template<typename Properties>
BaseDerivationRef<Properties>& smtrat::cadcells::datastructures::SampledDerivation< Properties
>::base ( )  [inline]
```

15.438.3.2 base() [2/2]

```
template<typename Properties>
BaseDerivationRef<Properties>& smtrat::cadcells::datastructures::SampledDerivation< Properties
>::base ( ) const  [inline]
```

15.438.3.3 cell()

```
template<typename Properties>
const DelineationInterval& smtrat::cadcells::datastructures::SampledDerivation< Properties >↩
::cell ( ) const  [inline]
```

15.438.3.4 contains()

```
template<typename Properties>
template<typename P >
bool smtrat::cadcells::datastructures::SampledDerivation< Properties >::contains (
            const P & property ) const  [inline]
```

15.438.3.5 delin()

```
template<typename Properties>
Delineation& smtrat::cadcells::datastructures::SampledDerivation< Properties >::delin ( )
[inline]
```

15.438.3.6 delineate_cell()

```
template<typename Properties>
void smtrat::cadcells::datastructures::SampledDerivation< Properties >::delineate_cell ( )
[inline]
```

Determines the cell w.r.t.

the delineation.

15.438.3.7 delineated() [1/2]

```
template<typename Properties>
DelineatedDerivationRef<Properties>& smtrat::cadcells::datastructures::SampledDerivation<
Properties >::delineated ( )  [inline]
```

15.438.3.8 delineated() [2/2]

```
template<typename Properties>
DelineatedDerivationRef<Properties>& smtrat::cadcells::datastructures::SampledDerivation<
Properties >::delineated ( ) const [inline]
```

15.438.3.9 insert()

```
template<typename Properties>
template<typename P >
void smtrat::cadcells::datastructures::SampledDerivation< Properties >::insert (
            P property ) [inline]
```

15.438.3.10 level()

```
template<typename Properties>
size_t smtrat::cadcells::datastructures::SampledDerivation< Properties >::level ( ) const
[inline]
```

15.438.3.11 main_var()

```
template<typename Properties>
carl::Variable smtrat::cadcells::datastructures::SampledDerivation< Properties >::main_var ( )
const [inline]
```

15.438.3.12 main_var_sample()

```
template<typename Properties>
const RAN& smtrat::cadcells::datastructures::SampledDerivation< Properties >::main_var_sample
( ) const [inline]
```

15.438.3.13 merge_with()

```
template<typename Properties>
void smtrat::cadcells::datastructures::SampledDerivation< Properties >::merge_with (
            const SampledDerivation< Properties > & other ) [inline]
```

15.438.3.14 polys()

```
template<typename Properties>
PolyPool& smtrat::cadcells::datastructures::SampledDerivation< Properties >::polys ( ) [inline]
```

15.438.3.15 proj()

```
template<typename Properties>
Projections& smtrat::cadcells::datastructures::SampledDerivation< Properties >::proj ( )
[inline]
```

15.438.3.16 properties()

```
template<typename Properties>
template<typename P >
const PropertiesTSet<P>& smtrat::cadcells::datastructures::SampledDerivation< Properties >↩
::properties ( ) const [inline]
```

15.438.3.17 sample()

```
template<typename Properties>
const Assignment& smtrat::cadcells::datastructures::SampledDerivation< Properties >::sample (
) const [inline]
```

15.438.3.18 underlying() `template<typename Properties>`

```
DerivationRef<Properties>& smtrat::cadcells::datastructures::SampledDerivation< Properties
>::underlying ( ) [inline]
```

15.438.3.19 underlying_sample() `template<typename Properties>`

```
const Assignment& smtrat::cadcells::datastructures::SampledDerivation< Properties >::underlying↩
_sample ( ) const [inline]
```

15.439 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

15.439.1 Member Function Documentation

15.439.1.1 operator()() `template<typename T >`

```
constexpr bool smtrat::SampledDerivationRefCompare::operator() (
            const cadcells::datastructures::SampledDerivationRef< T > & a,
            const cadcells::datastructures::SampledDerivationRef< T > & b ) const [inline],
[constexpr]
```

15.440 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 isConsistent () const

15.440.1 Member Function Documentation

15.440.1.1 addNewSample() `template<typename Iterator, typename Comparator>`

```
void smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::addNewSample (
            Iterator it ) [inline]
```

15.440.1.2 addNewSamples()

```
template<typename Iterator, typename Comparator>
template<typename InputIt >
void smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::addNewSamples (
            InputIt begin,
            InputIt end )  [inline]
```

15.440.1.3 assign()

```
template<typename Iterator, typename Comparator>
template<typename InputIt >
void smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::assign (
            InputIt begin,
            InputIt end )  [inline]
```

15.440.1.4 begin()

```
template<typename Iterator, typename Comparator>
auto smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::begin ( ) const  [inline]
```

15.440.1.5 cleanup()

```
template<typename Iterator, typename Comparator>
template<typename Filter >
void smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::cleanup (
            Filter && f )  [inline]
```

15.440.1.6 clear()

```
template<typename Iterator, typename Comparator>
void smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::clear ( )  [inline]
```

15.440.1.7 empty()

```
template<typename Iterator, typename Comparator>
bool smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::empty ( ) const  [inline]
```

15.440.1.8 end()

```
template<typename Iterator, typename Comparator>
auto smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::end ( ) const  [inline]
```

15.440.1.9 getNextSample()

```
template<typename Iterator, typename Comparator>
Iterator smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::getNextSample ( )  [inline]
```

15.440.1.10 isConsistent()

```
template<typename Iterator, typename Comparator>
bool smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::isConsistent ( ) const  [inline]
```

15.440.1.11 removeNextSample()

```
template<typename Iterator, typename Comparator>
Iterator smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::removeNextSample ( )  [inline]
```

15.441 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)

15.441.1 Member Function Documentation

15.441.1.1 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]
```

15.442 smtrat::sampling< SamplingAlgorithm::LOWER_UPPER_BETWEEN_SAMPLING > 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 > &derivationsSet)

15.442.1 Member Function Documentation

15.442.1.1 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]
```

15.443 smtrat::SATModule< Settings > Class Template Reference

Implements a module performing DPLL style SAT checking.

```
#include <SATModule.h>
```

Public Types

- typedef Settings SettingsType
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::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 (EvalRationalMap &_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 &="") 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 smtrat::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 _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)

  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 &_branching↩Value) 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 _preferLeft↩Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩::vector< FormulaT >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩CaseWeak=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.

- 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

15.443.1 Detailed Description

template<class Settings>
class smtrat::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.

15.443.2 Member Typedef Documentation

15.443.2.1 SettingsType `template<class Settings >`

```
typedef Settings smtrat::SATModule< Settings >::SettingsType
```

15.443.3 Member Enumeration Documentation

15.443.3.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

15.443.4 Constructor & Destructor Documentation

15.443.4.1 SATModule() `template<class Settings >`

```
smtrat::SATModule< Settings >::SATModule (
            const ModuleInput * _formula,
            Conditionals & _foundAnswer,
            Manager *const _manager = nullptr )
```

Constructs a SATModule.

Parameters

_type	The type of this module being SATModule.
_formula	The formula passed to this module, called received formula.
_settings	[Not yet used.]
_foundAnswer	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.

15.443.4.2 ~SATModule() `template<class Settings >`

```
smtrat::SATModule< Settings >::~SATModule ( )
```

Destructs this SATModule.

15.443.5 Member Function Documentation

15.443.5.1 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.

15.443.5.2 addBooleanAssignments()

```
template<class Settings >
void smtrat::SATModule< Settings >::addBooleanAssignments (
            EvalRationalMap & _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

_rationalAssignment	The assignments to add the Boolean assignments to.

15.443.5.3 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

_constraint	The constraint to add.

Reimplemented from smtrat::Module.

15.443.5.4 addConstraintToInform_()

```
template<class Settings >
void smtrat::SATModule< Settings >::addConstraintToInform_ (
            const FormulaT & _formula )  [inline]
```

15.443.5.5 addCore()

```
template<class Settings >
bool smtrat::SATModule< 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.

15.443.5.6 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.443.5.7 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.

15.443.5.8 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.

15.443.5.9 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.

15.443.5.10 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.

15.443.5.11 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

_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.

15.443.5.12 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

_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.

15.443.5.13 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns All satisfying assignments, if existent.

15.443.5.14 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.

15.443.5.15 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.

15.443.5.16 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.

15.443.5.17 branchAt() [1/6]

```
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]
```

15.443.5.18 branchAt() [2/6]

```
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]
```

15.443.5.19 branchAt() [3/6]

```
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]
```

15.443.5.20 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.443.5.21 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            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]
```

15.443.5.22 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

15.443.5.23 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in smtrat::PModule.

15.443.5.24 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.

15.443.5.25 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
std::vector< FormulaSetT >::const_iterator _infsubset,
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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.443.5.26 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.

15.443.5.27 cleanModel()

void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]

Substitutes variable occurrences by its model value in the model values of other variables.

15.443.5.28 cleanUpAfterOptimizing() template<class Settings >

```
void smtrat::SATModule< Settings >::cleanUpAfterOptimizing (
const std::vector< Minisat::CRef > & _excludedAssignments )
```

15.443.5.29 clearLemmas()

void smtrat::Module::clearLemmas ( ) [inline], [inherited]

Deletes all yet found lemmas.

15.443.5.30 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.443.5.31 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.443.5.32 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.443.5.33 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

15.443.5.34 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.

15.443.5.35 collectStats()

```
template<class Settings >
void smtrat::SATModule< Settings >::collectStats ( )
```

Collects the taken statistics.

15.443.5.36 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.443.5.37 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraintsToInform ( ) const [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.443.5.38 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

15.443.5.39 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)

15.443.5.40 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

_constraint	The constraint to remove from internal data structures.

15.443.5.41 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.443.5.42 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.443.5.43 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.443.5.44 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceived↩
VariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.443.5.45 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBest↩
Origin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.443.5.46 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.

15.443.5.47 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.

15.443.5.48 freeSplittingVariable() static void smtrat::Module::freeSplittingVariable (const FormulaT & *_splittingVariable*) [inline], [static], [inherited]

15.443.5.49 generateTrivialInfeasibleSubset() void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]

Stores the trivial infeasible subset being the set of received formulas.

15.443.5.50 getBackendsAllModels() void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]

Stores all models of a backend in the list of all models of this module.

15.443.5.51 getBackendsModel() void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]

15.443.5.52 getInfeasibleSubsets() [1/2] void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]

Copies the infeasible subsets of the passed formula.

15.443.5.53 getInfeasibleSubsets() [2/2] std::vector< FormulaSetT > smtrat::Module::getInfeasible↩ Subsets (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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.443.5.54 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.443.5.55 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.

15.443.5.56 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.443.5.57 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.443.5.58 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.443.5.59 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 reveived formula, if the Boolean is true.

Reimplemented in smtrat::PModule.

15.443.5.60 hasLemmas() bool smtrat::Module::hasLemmas ( ) [inline], [inherited]

Checks whether this module or any of its backends provides any lemmas.

15.443.5.61 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).

15.443.5.62 id() std::size_t smtrat::Module::id ( ) const [inline], [inherited]

Returns A unique ID to identify this module instance.

15.443.5.63 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.

15.443.5.64 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.443.5.65 informBackends()

```
void smtrat::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.

15.443.5.66 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, and smtrat::ICPModule< ICPSettings1 >.

15.443.5.67 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.443.5.68 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::FPPModule< Settings >.

15.443.5.69 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.443.5.70 isLemmaLevel() `bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.443.5.71 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().

15.443.5.72 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.

15.443.5.73 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.443.5.74 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.443.5.75 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.

15.443.5.76 moduleName() `virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]`

Returns The name of the given module type as name.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::EQPreprocessingModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::VSModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::GBModule< Settings >, smtrat::IntEqModule< Setti smtrat::CubeLIAModule< Settings >, smtrat::CurryModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IncWidthModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::BEModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::SplitSOSModule< Settings >.

15.443.5.77 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

15.443.5.78 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

15.443.5.79 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula. TODO: disable this method

15.443.5.80 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula.

15.443.5.81 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to the backends of this module.

15.443.5.82 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to this module.

15.443.5.83 print() [1/2]

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.443.5.84 print() [2/2]

`template<class Settings > void smtrat::SATModule< Settings >::print (`

```
std::ostream & _out = std::cout,
const std::string & _init = "" ) const
```

Prints everything.

Parameters

_out	The output stream where the answer should be printed.
_init	The line initiation.

15.443.5.85 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

_out	The stream to print the assignment on.

15.443.5.86 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

_out	The output stream where the answer should be printed.
_init	The line initiation.

15.443.5.87 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

_out	The output stream where the answer should be printed.
_init	The line initiation.

15.443.5.88 printClause() [1/2] template<class Settings >

```
void smtrat::SATModule< Settings >::printClause (
            const Minisat::vec< Minisat::Lit > & _clause,
            bool _withAssignment = false,
            std::ostream & _out = std::cout,
            const std::string & _init = "" ) const
```

Prints the clause resulting from the given vector of literals.

Parameters

_clause	The reference of the clause.
_withAssignment	A flag indicating if true, that the assignments should be printed too.
_out	The output stream where the answer should be printed.
_init	The prefix of each printed line.

15.443.5.89 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

_clause	The reference of the clause.
_withAssignment	A flag indicating if true, that the assignments should be printed too.
_out	The output stream where the answer should be printed.
_init	The prefix of each printed line.

15.443.5.90 printClauseInformation() template<class Settings >

```
void smtrat::SATModule< Settings >::printClauseInformation (
            std::ostream & _out = std::cout,
            const std::string & _init = "" ) const
```

Prints the clause information.

Parameters

_out	The output stream where the answer should be printed.
_init	The line initiation.

15.443.5.91 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 _from = 0,
bool _withAssignment = false,
bool _onlyNotSatisfied = false ) const
```

Prints all given clauses.

Parameters

_clauses	The clauses to print.
_name	The name of the clauses to print. (e.g. learnts, clauses, ..)
_out	The output stream where the answer should be printed.
_init	The prefix of each printed line.
_from	The position of the first clause to print within the given vector of clauses.
_withAssignment	A flag indicating if true, that the assignments should be printed too.

15.443.5.92 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

_out	The output stream where the answer should be printed.
_init	The line initiation.

15.443.5.93 printCurrentAssignment()

```
template<class Settings >
void smtrat::SATModule< Settings >::printCurrentAssignment (
            std::ostream & _out = std::cout,
            const std::string & _init = "" ) const
```

Prints the current assignment of the SAT solver.

Parameters

_out	The output stream where the answer should be printed.
_init	The line initiation.

15.443.5.94 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

_out	The output stream where the answer should be printed.
_init	The line initiation.

15.443.5.95 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

_out	The output stream where the answer should be printed.
_init	The line initiation.

15.443.5.96 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
            const std::string & _initiation = "***" ) const  [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.443.5.97 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

_out	The output stream where the answer should be printed.
_init	The line initiation.

15.443.5.98 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

_out	The stream to print the assignment on.

15.443.5.99 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.443.5.100 printPropagatedLemmas()

```
template<class Settings >
void smtrat::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

_out	The output stream where the answer should be printed.
_init	The line initiation.

15.443.5.101 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.443.5.102 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.

15.443.5.103 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

15.443.5.104 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

15.443.5.105 receivedVariable()

`bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

15.443.5.106 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.

15.443.5.107 removeCore()

`template<class Settings > void smtrat::SATModule< Settings >::removeCore ( ModuleInput::const_iterator _subformula ) [virtual]`

Removes everything related to the given sub-formula of the received formula.

Parameters

_subformula	The sub formula of the received formula to remove.

Reimplemented from smtrat::Module.

15.443.5.108 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.443.5.109 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]
```

15.443.5.110 removeUpperBoundOnMinimal()

```
template<class Settings >
void smtrat::SATModule< Settings >::removeUpperBoundOnMinimal ( )
```

15.443.5.111 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.443.5.112 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.443.5.113 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]
```

Reimplemented in smtrat::PModule.

15.443.5.114 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

_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 in smtrat::PModule.

15.443.5.115 setId() `void smtrat::Module::setId ( std::size_t _id ) [inline], [inherited]`

Sets this modules unique ID to identify itself.

Parameters

↩ _id	The id to set this module's id to.

15.443.5.116 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.

15.443.5.117 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.

15.443.5.118 splitUnequalConstraint() `void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.443.5.119 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.443.5.120 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.

15.443.5.121 updateInfeasibleSubset() `template<class Settings > void smtrat::SATModule< Settings >::updateInfeasibleSubset ( )`

Updates the infeasible subset found by the SATModule, if the received formula is unsatisfiable.

15.443.5.122 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.443.5.123 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.

15.443.5.124 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).

15.443.6 Friends And Related Function Documentation

15.443.6.1 VarSchedulerBase `template<class Settings > friend class VarSchedulerBase [friend]`

15.443.6.2 VarSchedulerMcsatBase `template<class Settings > friend class VarSchedulerMcsatBase [friend]`

15.443.7 Field Documentation

15.443.7.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.443.7.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.443.7.3 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.

15.443.7.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.443.7.5 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

15.443.7.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.443.7.7 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.

15.443.7.8 mFirstUncheckedReceivedSubformula `ModuleInput::const_iterator smtrat::Module::m↩ FirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

15.443.7.9 mFoundAnswer `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.443.7.10 mFullCheck `bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.443.7.11 mInfeasibleSubsets `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.443.7.12 mInformedConstraints `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.443.7.13 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.

15.443.7.14 mModel `Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.443.7.15 mModelComputed `bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.443.7.16 mNumOfBranchVarsToStore `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.443.7.17 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.

15.443.7.18 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

15.443.7.19 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.443.7.20 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.443.7.21 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.

15.443.7.22 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

15.443.7.23 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).

15.443.7.24 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.444 smtrat::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

15.444.1 Constructor & Destructor Documentation

15.444.1.1 ScriptParser() template<typename Callee>

```
smtrat::parser::ScriptParser< Callee >::ScriptParser (
            InstructionHandler & h,
            Theories & theories,
            Callee & callee )  [inline]
```

15.444.2 Member Function Documentation

15.444.2.1 startFunctionDefinition() template<typename Callee>

```
void smtrat::parser::ScriptParser< Callee >::startFunctionDefinition ( )  [inline]
```

15.444.3 Field Documentation

15.444.3.1 attribute template<typename Callee>

```
AttributeParser smtrat::parser::ScriptParser< Callee >::attribute
```

15.444.3.2 callee template<typename Callee>

```
Callee& smtrat::parser::ScriptParser< Callee >::callee
```

15.444.3.3 command template<typename Callee>

```
qi::rule<Iterator, Skipper> smtrat::parser::ScriptParser< Callee >::command
```

15.444.3.4 decimal template<typename Callee>

```
DecimalParser smtrat::parser::ScriptParser< Callee >::decimal
```

15.444.3.5 errorHandler template<typename Callee>

```
px::function<ErrorHandler> smtrat::parser::ScriptParser< Callee >::errorHandler
```

15.444.3.6 functionDefinition template<typename Callee>

qi::rule<Iterator, Skipper> smtrat::parser::ScriptParser< Callee >::functionDefinition

15.444.3.7 functionDefinitionArg template<typename Callee>

qi::rule<Iterator, types::VariableType(), Skipper> smtrat::parser::ScriptParser< Callee >↩
::functionDefinitionArg

15.444.3.8 handler template<typename Callee>

InstructionHandler& smtrat::parser::ScriptParser< Callee >::handler

15.444.3.9 keyword template<typename Callee>

KeywordParser smtrat::parser::ScriptParser< Callee >::keyword

15.444.3.10 logic template<typename Callee>

LogicParser smtrat::parser::ScriptParser< Callee >::logic

15.444.3.11 main template<typename Callee>

qi::rule<Iterator, Skipper> smtrat::parser::ScriptParser< Callee >::main

15.444.3.12 numeral template<typename Callee>

NumeralParser smtrat::parser::ScriptParser< Callee >::numeral

15.444.3.13 qeQuery template<typename Callee>

QEParser smtrat::parser::ScriptParser< Callee >::qeQuery

15.444.3.14 sort template<typename Callee>

SortParser smtrat::parser::ScriptParser< Callee >::sort

15.444.3.15 sortedvariable template<typename Callee>

SortedVariableParser smtrat::parser::ScriptParser< Callee >::sortedvariable

15.444.3.16 state template<typename Callee>

ParserState smtrat::parser::ScriptParser< Callee >::state

15.444.3.17 string `template<typename Callee>`

`StringParser smtrat::parser::ScriptParser< Callee >::string`

15.444.3.18 symbol `template<typename Callee>`

`SymbolParser smtrat::parser::ScriptParser< Callee >::symbol`

15.444.3.19 term `template<typename Callee>`

`TermParser smtrat::parser::ScriptParser< Callee >::term`

15.444.3.20 theories `template<typename Callee>`

`Theories& smtrat::parser::ScriptParser< Callee >::theories`

15.445 smtrat::parser::ParserState::ScriptScope Struct Reference

`#include <ParserState.h>`

Public Member Functions

- ScriptScope (const ParserState &state)
- void discharge (ParserState &state)

15.445.1 Constructor & Destructor Documentation

15.445.1.1 ScriptScope()

```
smtrat::parser::ParserState::ScriptScope::ScriptScope (
    const ParserState & state ) [inline]
```

15.445.2 Member Function Documentation

15.445.2.1 discharge()

```
void smtrat::parser::ParserState::ScriptScope::discharge (
    ParserState & state ) [inline]
```

15.446 smtrat::mcsat::onecellcad::firstlevel::section Struct Reference

`#include <first_level.h>`

Data Fields

- std::vector< Poly > current_level
- std::vector< Poly > lower_level
- indexed_root point_defining_poly
- RAN point
- std::set< Poly > others

15.446.1 Field Documentation

15.446.1.1 current_level

`std::vector<Poly> smtrat::mcsat::onecellcad::firstlevel::section↩ ::current_level`

15.446.1.2 lower_level

`std::vector<Poly> smtrat::mcsat::onecellcad::firstlevel::section::lower↩ _level`

15.446.1.3 others `std::set<Poly> smtrat::mcsat::onecellcad::firstlevel::section::others`

15.446.1.4 point `RAN smtrat::mcsat::onecellcad::firstlevel::section::point`

15.446.1.5 point_defining_poly `indexed_root smtrat::mcsat::onecellcad::firstlevel::section↩ ::point_defining_poly`

15.447 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]).

15.447.1 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^{k-1} -> 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).

15.447.2 Field Documentation

15.447.2.1 cachedPoint `RAN smtrat::onecellcad::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]).

15.447.2.2 poly `MultiPoly smtrat::onecellcad::recursive::Section::poly`

15.448 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.

15.448.1 Detailed Description

Represent a cell's (closed-interval-boundary) component along th k-th axis.

A section is a function f: algReal^{k-1} -> 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.

15.448.2 Field Documentation

15.448.2.1 boundFunction `MultivariateRootT smtrat::mcsat::onecellcad::Section::boundFunction`

15.448.2.2 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.

15.449 smtrat::mcsat::onecellcad::levelwise::SectionHeuristic1 Struct Reference

`#include <Explanation.h>`

Static Public Attributes

- static constexpr int sectionHeuristic = 1

15.449.1 Field Documentation

15.449.1.1 sectionHeuristic `constexpr int smtrat::mcsat::onecellcad::levelwise::SectionHeuristic1↩ ::sectionHeuristic = 1 [static], [constexpr]`

15.450 smtrat::mcsat::onecellcad::levelwise::SectionHeuristic2 Struct Reference

`#include <Explanation.h>`

Static Public Attributes

- static constexpr int sectionHeuristic = 2

15.450.1 Field Documentation

15.450.1.1 sectionHeuristic `constexpr int smtrat::mcsat::onecellcad::levelwise::SectionHeuristic2↩ ::sectionHeuristic = 2 [static], [constexpr]`

15.451 smtrat::mcsat::onecellcad::levelwise::SectionHeuristic3 Struct Reference

`#include <Explanation.h>`

Static Public Attributes

- static constexpr int sectionHeuristic = 3

15.451.1 Field Documentation

15.451.1.1 sectionHeuristic `constexpr int smtrat::mcsat::onecellcad::levelwise::SectionHeuristic3↩ ::sectionHeuristic = 3 [static], [constexpr]`

15.452 smtrat::mcsat::onecellcad::firstlevel::sector Struct Reference

`#include <first_level.h>`

Data Fields

- std::vector< Poly > current_level
- std::vector< Poly > lower_level
- std::optional< indexed_root > left_defining_poly
- std::optional< indexed_root > right_defining_poly
- std::optional< RAN > left
- std::optional< RAN > right
- std::set< Poly > others_left
- std::set< Poly > others_right

15.452.1 Field Documentation

15.452.1.1 current_level

`std::vector<Poly> smtrat::mcsat::onecellcad::firstlevel::sector::current_level`

15.452.1.2 left

`std::optional<RAN> smtrat::mcsat::onecellcad::firstlevel::sector::left`

15.452.1.3 left_defining_poly

`std::optional<indexed_root> smtrat::mcsat::onecellcad::firstlevel::sector::left_defining_poly`

15.452.1.4 lower_level

`std::vector<Poly> smtrat::mcsat::onecellcad::firstlevel::sector::lower_level`

15.452.1.5 others_left

`std::set<Poly> smtrat::mcsat::onecellcad::firstlevel::sector::others_left`

15.452.1.6 others_right

`std::set<Poly> smtrat::mcsat::onecellcad::firstlevel::sector::others_right`

15.452.1.7 right

`std::optional<RAN> smtrat::mcsat::onecellcad::firstlevel::sector::right`

15.452.1.8 right_defining_poly

`std::optional<indexed_root> smtrat::mcsat::onecellcad::firstlevel::sector::right_defining_poly`

15.453 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.

15.453.1 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: algReal^{k-1} -> (algReal,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.

15.453.2 Field Documentation

15.453.2.1 highBound `std::optional<Section> smtrat::mcsat::onecellcad::Sector::highBound`

A std:nullopt highBound represents infinity.

15.453.2.2 lowBound `std::optional<Section> smtrat::mcsat::onecellcad::Sector::lowBound`

A std::nullopt lowBound represents negative infinity.

15.454 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

15.454.1 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: algReal^{k-1} -> (algReal,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.

15.454.2 Member Function Documentation

15.454.2.1 isHighBoundInfty() `bool smtrat::onecellcad::recursive::Sector::isHighBoundInfty ( ) const [inline]`

15.454.2.2 isLowBoundNegInfty() `bool smtrat::onecellcad::recursive::Sector::isLowBoundNegInfty ( ) const [inline]`

15.454.3 Field Documentation

15.454.3.1 highBound `std::optional<Section> smtrat::onecellcad::recursive::Sector::highBound`

15.454.3.2 lowBound `std::optional<Section> smtrat::onecellcad::recursive::Sector::lowBound`

15.455 smtrat::mcsat::onecellcad::levelwise::SectorHeuristic1 Struct Reference

```
#include <Explanation.h>
```

Static Public Attributes

- static constexpr int sectorHeuristic = 1

15.455.1 Field Documentation

15.455.1.1 sectorHeuristic `constexpr int smtrat::mcsat::onecellcad::levelwise::SectorHeuristic1↩ ::sectorHeuristic = 1 [static], [constexpr]`

15.456 smtrat::mcsat::onecellcad::levelwise::SectorHeuristic2 Struct Reference

`#include <Explanation.h>`

Static Public Attributes

- static constexpr int sectorHeuristic = 2

15.456.1 Field Documentation

15.456.1.1 sectorHeuristic `constexpr int smtrat::mcsat::onecellcad::levelwise::SectorHeuristic2↩ ::sectorHeuristic = 2 [static], [constexpr]`

15.457 smtrat::mcsat::onecellcad::levelwise::SectorHeuristic3 Struct Reference

`#include <Explanation.h>`

Static Public Attributes

- static constexpr int sectorHeuristic = 3

15.457.1 Field Documentation

15.457.1.1 sectorHeuristic `constexpr int smtrat::mcsat::onecellcad::levelwise::SectorHeuristic3↩ ::sectorHeuristic = 3 [static], [constexpr]`

15.458 smtrat::mcsat::SequentialAssignment< Backends > Struct Template Reference

`#include <SequentialAssignment.h>`

15.459 smtrat::mcsat::SequentialExplanation< Backends > Struct Template Reference

`#include <SequentialExplanation.h>`

15.460 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.

15.460.1 Detailed Description

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

15.460.2 Constructor & Destructor Documentation

15.460.2.1 Settings()

```
delta::Settings::Settings (
    const std::string & executable ) [inline]
```

Constructor.

Parameters

executable	Name of this binary.

15.460.3 Member Function Documentation

15.460.3.1 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 of it has a different type, exceptions may be thrown.

Parameters

s	Name of the option.

Returns Value of the option as type T.

15.460.3.2 has()

```
bool delta::Settings::has (
            const std::string & s ) const  [inline]
```

Checks if there is an option with this name.

Parameters

s	Name of the option.

Returns If option was set.

15.460.3.3 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

s	Name of the option.

Returns Whether it's value was set as the default.

15.460.3.4 load()

```
bool delta::Settings::load (
    int argc,
    char * argv[] ) [inline]
```

Load options from command line.

Parameters

argc	Number of arguments.
argv	Content of arguments.

15.460.3.5 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

s	Name of the option.
t	New value.

15.460.4 Friends And Related Function Documentation

15.460.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const Settings & s ) [friend]
```

Streaming operator.

Parameters

os	Output stream.
s	Settings.

Returns os.

15.461 benchmax::settings::Settings Struct Reference

Generic class to manage runtime settings.

```
#include <Settings.h>
```

15.461.1 Detailed Description

Generic class to manage runtime settings.

Essentially stores all (named) settings in a map<string,any> and retrieves settings classes when requested.

15.462 smtrat::settings::Settings Struct Reference

```
#include <Settings.h>
```

15.463 smtrat::SettingsComponents Class Reference

```
#include <SettingsComponents.h>
```

Public Member Functions

- void add (std::function< void(SettingsParser &)> &&f)
- void add_to_parser (SettingsParser &parser) const

15.463.1 Member Function Documentation

15.463.1.1 add()

```
void smtrat::SettingsComponents::add (
    std::function< void(SettingsParser &)> && f ) [inline]
```

15.463.1.2 add_to_parser()

```
void smtrat::SettingsComponents::add_to_parser (
    SettingsParser & parser ) const [inline]
```

15.464 benchmax::SettingsParser Class Reference

Generic class to manage settings parsing using boost::program_options.

```
#include <SettingsParser.h>
```

15.464.1 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 registed finalize() has to be called to construct the full option description.

15.465 smtrat::SettingsParser Class Reference

```
#include <SettingsParser.h>
```

15.466 smtrat::CNFerModule::SettingsType Struct Reference

```
#include <CNFerModule.h>
```

Static Public Attributes

- static constexpr auto moduleName = "CNFerModule"

15.466.1 Field Documentation

15.466.1.1 moduleName

```
constexpr auto smtrat::CNFerModule::SettingsType::moduleName = "CN↩
FerModule" [static], [constexpr]
```

15.467 smtrat::parser::SExpressionParser Struct Reference

```
#include <SExpression.h>
```

Public Member Functions

- SExpressionParser ()

Data Fields

- SpecConstantParser specconstant
- SymbolParser symbol
- KeywordParser keyword
- qi::rule< Iterator, SExpression< types::ConstType >), Skipper > main

15.467.1 Constructor & Destructor Documentation

15.467.1.1 SExpressionParser() `smtrat::parser::SExpressionParser::SExpressionParser ( ) [inline]`

15.467.2 Field Documentation

15.467.2.1 keyword `KeywordParser smtrat::parser::SExpressionParser::keyword`

15.467.2.2 main `qi::rule<Iterator, SExpression<types::ConstType>), Skipper> smtrat::parser↩ ::SExpressionParser::main`

15.467.2.3 specconstant `SpecConstantParser smtrat::parser::SExpressionParser::specconstant`

15.467.2.4 symbol `SymbolParser smtrat::parser::SExpressionParser::symbol`

15.468 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.

15.468.1 Constructor & Destructor Documentation

15.468.1.1 SExpressionSequence() [1/2]

```
template<typename T>
smtrat::parser::SExpressionSequence< T >::SExpressionSequence (
            const std::vector< SExpression< T >> & v ) [inline]
```

15.468.1.2 SExpressionSequence() [2/2]

```
template<typename T>
smtrat::parser::SExpressionSequence< T >::SExpressionSequence (
            std::vector< SExpression< T >> && v ) [inline]
```

15.468.2 Field Documentation

15.468.2.1 elements `T std::vector< T >::elements [inherited]`

STL member.

15.469 smtrat::ShortFormulaEncoder Class Reference

```
#include <ShortFormulaEncoder.h>
```

Public Member Functions

- ShortFormulaEncoder ()
- bool canEncode (const ConstraintT &constraint)
- Rational encodingSize (const ConstraintT &constraint)
- std::string name ()
- boost::optional< FormulaT > encode (const ConstraintT &constraint)

  Encodes an arbitrary constraint.

Data Fields

- std::size_t problem_size

Protected Member Functions

- boost::optional< FormulaT > doEncode (const ConstraintT &constraint)
- FormulaT generateVarChain (const std::set< carl::Variable > &vars, carl::FormulaType type)

15.469.1 Constructor & Destructor Documentation

15.469.1.1 ShortFormulaEncoder() `smtrat::ShortFormulaEncoder::ShortFormulaEncoder ( ) [inline]`

15.469.2 Member Function Documentation

15.469.2.1 canEncode() `bool smtrat::ShortFormulaEncoder::canEncode ( const ConstraintT & constraint ) [virtual]`

Implements smtrat::PseudoBoolEncoder.

15.469.2.2 doEncode() `boost::optional< FormulaT > smtrat::ShortFormulaEncoder::doEncode ( const ConstraintT & constraint ) [protected], [virtual]`

Implements smtrat::PseudoBoolEncoder.

15.469.2.3 encode() `boost::optional< FormulaT > smtrat::PseudoBoolEncoder::encode ( const ConstraintT & constraint ) [inherited]`

Encodes an arbitrary constraint.

Returns encoded formula

15.469.2.4 encodingSize() `Rational smtrat::ShortFormulaEncoder::encodingSize ( const ConstraintT & constraint ) [virtual]`

Reimplemented from smtrat::PseudoBoolEncoder.

15.469.2.5 generateVarChain()

```
FormulaT smtrat::PseudoBoolEncoder::generateVarChain (
    const std::set< carl::Variable > & vars,
    carl::FormulaType type ) [protected], [inherited]
```

15.469.2.6 name()

```
std::string smtrat::ShortFormulaEncoder::name ( ) [inline], [virtual]
```

Reimplemented from smtrat::PseudoBoolEncoder.

15.469.3 Field Documentation

15.469.3.1 problem_size

```
std::size_t smtrat::PseudoBoolEncoder::problem_size [inherited]
```

15.470 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)

15.470.1 Constructor & Destructor Documentation

15.470.1.1 simple_parser()

```
benchmax::simple_parser::simple_parser (
    const std::string & content ) [inline]
```

15.470.2 Member Function Documentation

15.470.2.1 expect()

```
bool benchmax::simple_parser::expect (
    const char c ) [inline]
```

15.470.2.2 expect_end()

```
bool benchmax::simple_parser::expect_end ( ) [inline]
```

15.470.2.3 read_until()

```
std::optional<std::string> benchmax::simple_parser::read_until (
    const char c ) [inline]
```

15.470.2.4 read_until_whitespace()

```
std::optional<std::string> benchmax::simple_parser::read_↩
until_whitespace ( ) [inline]
```

15.470.2.5 read_until_whitespace_or()

```
std::optional<std::string> benchmax::simple_parser::read_↩
until_whitespace_or (
    const std::initializer_list< char > cs ) [inline]
```

15.470.2.6 skip_excluding()

```
void benchmax::simple_parser::skip_excluding (
    const char c ) [inline]
```

15.470.2.7 skip_whitespace()

```
void benchmax::simple_parser::skip_whitespace ( ) [inline]
```

15.471 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

15.471.1 Detailed Description

Helper functor for addSimpleSorts() method.

15.471.2 Constructor & Destructor Documentation

15.471.2.1 SimpleSortAdder()

```
smtrat::parser::Theories::SimpleSortAdder::SimpleSortAdder (
    qi::symbols< char, carl::Sort > & sorts ) [inline]
```

15.471.3 Member Function Documentation

15.471.3.1 operator()()

```
template<typename T >
void smtrat::parser::Theories::SimpleSortAdder::operator() (
    T * ) [inline]
```

15.471.4 Field Documentation

15.471.4.1 sorts

```
qi::symbols<char, carl::Sort>& smtrat::parser::Theories::SimpleSortAdder↩
::sorts
```

15.472 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

15.472.1 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.

15.472.2 Constructor & Destructor Documentation

15.472.2.1 SimpleSymbolParser() `smtrat::parser::SimpleSymbolParser::SimpleSymbolParser ( ) [inline]`

15.472.3 Field Documentation

15.472.3.1 main `qi::rule<Iterator, std::string(), Skipper> smtrat::parser::SimpleSymbol↩ Parser::main`

15.473 smtrat::expression::simplifier::Simplifier Class Reference

`#include <Simplifier.h>`

Public Member Functions

- const ExpressionContent ∗ operator() (const ExpressionContent ∗_ec) const

15.473.1 Member Function Documentation

15.473.1.1 operator()() `const ExpressionContent* smtrat::expression::simplifier::Simplifier↩ ::operator() (`

```
const ExpressionContent * _ec ) const [inline]
```

15.474 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

15.474.1 Member Function Documentation

15.474.1.1 operator()() `template<std::size_t chainID = std::tuple_size<SimplifierChain>::value - 1>`

```
const ExpressionContent* smtrat::expression::simplifier::SimplifierChainCaller< chainID >↩
::operator() (
            const ExpressionContent * _ec,
            const SimplifierChain & _chain ) const [inline]
```

15.474.2 Field Documentation

15.474.2.1 recurse `template<std::size_t chainID = std::tuple_size<SimplifierChain>::value - 1>`

`SimplifierChainCaller<chainID-1> smtrat::expression::simplifier::SimplifierChainCaller< chain↩ ID >::recurse`

15.475 smtrat::expression::simplifier::SimplifierChainCaller< 0 > Struct Template Reference

`#include <Simplifier.h>`

Public Member Functions

- const ExpressionContent ∗ operator() (const ExpressionContent ∗_ec, const SimplifierChain &_chain) const

15.475.1 Member Function Documentation

15.475.1.1 operator()()

```
const ExpressionContent* smtrat::expression::simplifier::SimplifierChainCaller<
0 >::operator() (
            const ExpressionContent * _ec,
            const SimplifierChain & _chain ) const  [inline]
```

15.476 smtrat::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

15.476.1 Member Function Documentation

15.476.1.1 operator()() [1/7]

```
const ExpressionContent* smtrat::expression::simplifier::Base↩
Simplifier::operator() (
            const BinaryExpression & expr ) const  [inline], [inherited]
```

15.476.1.2 operator()() [2/7]

```
const ExpressionContent* smtrat::expression::simplifier::Base↩
Simplifier::operator() (
            const carl::Variable & expr ) const  [inline], [inherited]
```

15.476.1.3 operator()() [3/7]

```
const ExpressionContent* smtrat::expression::simplifier::Base↩
Simplifier::operator() (
            const ExpressionContent * _ec ) const  [inline], [inherited]
```

15.476.1.4 operator()() [4/7]

```
const ExpressionContent* smtrat::expression::simplifier::Base↩
Simplifier::operator() (
            const ITEExpression & expr ) const  [inline], [inherited]
```

15.476.1.5 operator()() [5/7]

```
const ExpressionContent* smtrat::expression::simplifier::Base↩
Simplifier::operator() (
            const NaryExpression & expr ) const  [inline], [inherited]
```

15.476.1.6 operator()() [6/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const QuantifierExpression & expr ) const [inline], [inherited]
```

15.476.1.7 operator()() [7/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const UnaryExpression & expr ) const [inline], [inherited]
```

15.476.1.8 simplify() [1/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const BinaryExpression & ) const [inline], [protected], [virtual], [inherited]
```

15.476.1.9 simplify() [2/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const carl::Variable & ) const [inline], [protected], [virtual], [inherited]
```

15.476.1.10 simplify() [3/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const ITEExpression & ) const [inline], [protected], [virtual], [inherited]
```

15.476.1.11 simplify() [4/6]

```
const ExpressionContent* smtrat::expression::simplifier::SingletonSimplifier::simplify (
            const NaryExpression & expr ) const [inline], [virtual]
```

Reimplemented from smtrat::expression::simplifier::BaseSimplifier.

15.476.1.12 simplify() [5/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const QuantifierExpression & ) const [inline], [protected], [virtual], [inherited]
```

15.476.1.13 simplify() [6/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const UnaryExpression & ) const [inline], [protected], [virtual], [inherited]
```

Reimplemented in smtrat::expression::simplifier::NegationSimplifier.

15.477 smtrat::cad::sample_compare::size Struct Reference

```
#include <SampleComparator.h>
```

15.478 smtrat::parser::Skipper Struct Reference

```
#include <Common.h>
```

Public Member Functions

- Skipper ()

Data Fields

- boost::spirit::qi::rule< Iterator > main

15.478.1 Constructor & Destructor Documentation

15.478.1.1 Skipper() `smtrat::parser::Skipper::Skipper ( ) [inline]`

15.478.2 Field Documentation

15.478.2.1 main `boost::spirit::qi::rule<Iterator> smtrat::parser::Skipper::main`

15.479 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

15.479.1 Detailed Description This class is a `boost::spirit::qi` grammar that matches whitespaces and smtlib comments.

15.479.2 Constructor & Destructor Documentation

15.479.2.1 Skipper() `delta::Skipper::Skipper ( ) [inline]`

15.479.3 Field Documentation

15.479.3.1 main `qi::rule<Iterator> delta::Skipper::main`

15.480 benchmax::SlurmBackend Class Reference

Backend for the Slurm workload manager.

```
#include <SlurmBackend.h>
```

Public Member Functions

- bool suspendable () const
- void run (const Jobs &jobs, bool wait_for_termination)

  Run all tools on all benchmarks using Slurm.
- 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.

15.480.1 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.

15.480.2 Member Function Documentation

15.480.2.1 addResult()

```
void benchmax::Backend::addResult (
    const Tool * tool,
    const fs::path & file,
    BenchmarkResult && result ) [inline], [inherited]
```

Add a result.

15.480.2.2 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.

15.480.2.3 finalize()

```
virtual void benchmax::Backend::finalize ( ) [inline], [protected], [virtual],
[inherited]
```

Hook to allow for asynchronous backends to wait for jobs to terminate.

15.480.2.4 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.

15.480.2.5 process_results()

```
void benchmax::Backend::process_results (
    const Jobs & jobs,
    bool check_finished ) [inline], [inherited]
```

15.480.2.6 run()

```
void benchmax::SlurmBackend::run (
    const Jobs & jobs,
    bool wait_for_termination ) [inline]
```

Run all tools on all benchmarks using Slurm.

15.480.2.7 sanitize_results()

```
void benchmax::Backend::sanitize_results (
    const Jobs & jobs ) const [inline], [protected], [inherited]
```

15.480.2.8 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.

15.480.2.9 suspendable()

```
bool benchmax::SlurmBackend::suspendable ( ) const [inline]
```

15.480.2.10 write_results()

```
void benchmax::Backend::write_results (
    const Jobs & jobs ) const [inline], [protected], [inherited]
```

15.480.3 Field Documentation

15.480.3.1 mExpectedJobs

```
std::size_t benchmax::Backend::mExpectedJobs [protected], [inherited]
```

Number of jobs that should be run.

15.480.3.2 mFinishedJobs

```
std::atomic<std::size_t> benchmax::Backend::mFinishedJobs [protected],
[inherited]
```

Number of jobs that are finished.

15.480.3.3 mLastPercent

```
std::atomic<std::size_t> benchmax::Backend::mLastPercent [protected],
[inherited]
```

Percentage of finished jobs when madeProgress() was last called.

15.481 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.

15.481.1 Detailed Description

Settings for the Slurm backend.

15.481.2 Field Documentation

15.481.2.1 archive_log_file `std::string benchmax::settings::SlurmBackendSettings::archive_log_file`

Puts logs to some archive.

15.481.2.2 array_size `std::size_t benchmax::settings::SlurmBackendSettings::array_size`

Number of array jobs within one job.

15.481.2.3 keep_logs `bool benchmax::settings::SlurmBackendSettings::keep_logs`

Do not remove logs from file system if set to true.

15.481.2.4 sbatch_options `std::string benchmax::settings::SlurmBackendSettings::sbatch_options`

Additional options passed on to slurm.

15.481.2.5 slice_size `std::size_t benchmax::settings::SlurmBackendSettings::slice_size`

Size of one slice that is handled in one array job.

15.481.2.6 submission_delay `carl::settings::duration benchmax::settings::SlurmBackendSettings::submission_delay`

Delay between job submissions.

15.481.2.7 tmp_dir `std::string benchmax::settings::SlurmBackendSettings::tmp_dir`

Temporary directory for output files.

15.482 smtrat::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 (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

15.482.1 Constructor & Destructor Documentation

15.482.1.1 SMTLIBParser()

```
smtrat::parser::SMTLIBParser::SMTLIBParser (
    InstructionHandler & handler,
    bool queueInstructions ) [inline]
```

15.482.1.2 ~SMTLIBParser() `smtrat::parser::SMTLIBParser::~SMTLIBParser ( ) [inline]`

15.482.2 Member Function Documentation

15.482.2.1 add()

```
void smtrat::parser::SMTLIBParser::add (
    const types::TermType & t,
    bool isSoftFormula = false,
    Rational weight = Rational(1),
    const std::string id = std::string() ) [inline]
```

15.482.2.2 addObjective()

```
void smtrat::parser::SMTLIBParser::addObjective (
    const types::TermType & t,
    OptimizationType ot ) [inline]
```

15.482.2.3 callHandler()

```
template<typename Function , typename... Args>
void smtrat::parser::SMTLIBParser::callHandler (
    const Function & f,
    const Args &... args ) [inline]
```

15.482.2.4 check()

```
void smtrat::parser::SMTLIBParser::check ( ) [inline]
```

15.482.2.5 declareConst()

```
void smtrat::parser::SMTLIBParser::declareConst (
    const std::string & name,
    const carl::Sort & sort ) [inline]
```

15.482.2.6 declareFun()

```
void smtrat::parser::SMTLIBParser::declareFun (
    const std::string & name,
    const std::vector< carl::Sort > & args,
    const carl::Sort & sort ) [inline]
```

15.482.2.7 declareSort()

```
void smtrat::parser::SMTLIBParser::declareSort (
    const std::string & name,
    Integer arity ) [inline]
```

15.482.2.8 echo()

```
void smtrat::parser::SMTLIBParser::echo (
    const std::string & s ) [inline]
```

15.482.2.9 eliminateQuantifiers()

```
void smtrat::parser::SMTLIBParser::eliminateQuantifiers (
    const qe::QEQuery & q ) [inline]
```

15.482.2.10 exit()

```
void smtrat::parser::SMTLIBParser::exit ( ) [inline]
```

15.482.2.11 getAllModels()

```
void smtrat::parser::SMTLIBParser::getAllModels ( ) [inline]
```

15.482.2.12 getAssertions()

```
void smtrat::parser::SMTLIBParser::getAssertions ( ) [inline]
```

15.482.2.13 getAssignment()

```
void smtrat::parser::SMTLIBParser::getAssignment ( ) [inline]
```

15.482.2.14 getInfo()

```
void smtrat::parser::SMTLIBParser::getInfo (
    const std::string & key ) [inline]
```

15.482.2.15 getModel()

```
void smtrat::parser::SMTLIBParser::getModel ( ) [inline]
```

15.482.2.16 getObjectives()

```
void smtrat::parser::SMTLIBParser::getObjectives ( ) [inline]
```

15.482.2.17 getOption()

```
void smtrat::parser::SMTLIBParser::getOption (
    const std::string & key ) [inline]
```

15.482.2.18 getProof()

```
void smtrat::parser::SMTLIBParser::getProof ( ) [inline]
```

15.482.2.19 getUnsatCore()

```
void smtrat::parser::SMTLIBParser::getUnsatCore ( ) [inline]
```

15.482.2.20 getValue()

```
void smtrat::parser::SMTLIBParser::getValue (
    const std::vector< types::TermType > & vars ) [inline]
```

15.482.2.21 parse()

```
bool smtrat::parser::SMTLIBParser::parse (
    std::istream & in ) [inline]
```

15.482.2.22 pop()

```
void smtrat::parser::SMTLIBParser::pop (
    const Integer & n ) [inline]
```

15.482.2.23 push()

```
void smtrat::parser::SMTLIBParser::push (
    const Integer & n ) [inline]
```

15.482.2.24 reset()

```
void smtrat::parser::SMTLIBParser::reset ( ) [inline]
```

15.482.2.25 resetAssertions()

```
void smtrat::parser::SMTLIBParser::resetAssertions ( ) [inline]
```

15.482.2.26 setInfo()

```
void smtrat::parser::SMTLIBParser::setInfo (
    const Attribute & attribute ) [inline]
```

15.482.2.27 setLogic()

```
void smtrat::parser::SMTLIBParser::setLogic (
    const carl::Logic & name ) [inline]
```

15.482.2.28 setOption()

```
void smtrat::parser::SMTLIBParser::setOption (
    const Attribute & option ) [inline]
```

15.482.3 Field Documentation

15.482.3.1 handler

```
InstructionHandler& smtrat::parser::SMTLIBParser::handler
```

15.482.3.2 parser

```
ScriptParser<SMTLIBParser> smtrat::parser::SMTLIBParser::parser
```

15.482.3.3 queueInstructions

```
bool smtrat::parser::SMTLIBParser::queueInstructions
```

15.482.3.4 state

```
ParserState smtrat::parser::SMTLIBParser::state
```

15.482.3.5 theories

```
Theories smtrat::parser::SMTLIBParser::theories
```

15.483 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.

15.483.1 Detailed Description

Tool wrapper for SMT-RAT for SMT-LIB.

15.483.2 Constructor & Destructor Documentation

15.483.2.1 SMTRAT()

```
benchmax::SMTRAT::SMTRAT (
    const fs::path & binary,
    const std::string & arguments ) [inline]
```

New SMT-RAT tool.

15.483.3 Member Function Documentation

15.483.3.1 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.

15.483.3.2 attributeHash()

```
std::size_t benchmax::Tool::attributeHash ( ) const [inline], [inherited]
```

Hash of the attributes.

15.483.3.3 attributes()

```
const std::map<std::string,std::string>& benchmax::Tool::attributes ( )
const [inline], [inherited]
```

A set of attributes, for example compilation options.

15.483.3.4 binary()

```
fs::path benchmax::Tool::binary ( ) const [inline], [inherited]
```

Full path to the binary.

15.483.3.5 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.

15.483.3.6 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.

15.483.3.7 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.

15.483.3.8 getStatusFromOutput()

```
std::string benchmax::SMTRAT::getStatusFromOutput (
    const BenchmarkResult & result ) const [inline]
```

Try to parse memouts from stderr.

15.483.3.9 name() `std::string benchmax::Tool::name ( ) const [inline], [inherited]`

Common name of this tool.

15.483.3.10 parse_stats() `bool benchmax::SMTRAT::parse_stats ( BenchmarkResult & result ) const [inline]`

15.483.3.11 parseCommandline() `virtual std::optional<std::string> benchmax::Tool::parse↩ Commandline ( const std::string & cmdline ) const [inline], [virtual], [inherited]`

Compose commandline for this tool and the given input file.

15.483.3.12 resolveDependencies() `std::vector<std::string> benchmax::Tool::resolveDependencies ( ) const [inline], [inherited]`

Get dependencies of binary required to run it (via ldd)

15.483.4 Field Documentation

15.483.4.1 mArguments `std::string benchmax::Tool::mArguments [protected], [inherited]`

Command line arguments that should be passed to the binary.

15.483.4.2 mAttributes `std::map<std::string,std::string> benchmax::Tool::mAttributes [protected], [inherited]`

Attributes of the tool obtained by introspection of the binary.

15.483.4.3 mBinary `fs::path benchmax::Tool::mBinary [protected], [inherited]`

Path to the binary.

15.483.4.4 mName `std::string benchmax::Tool::mName [protected], [inherited]`

(Non-unique) identifier for the tool.

15.484 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.

15.484.1 Detailed Description

Tool wrapper for SMT-RAT for SMT-LIB.

15.484.2 Constructor & Destructor Documentation

15.484.2.1 SMTRAT_Analyzer()

```
benchmax::SMTRAT_Analyzer::SMTRAT_Analyzer (
    const fs::path & binary,
    const std::string & arguments ) [inline]
```

New SMT-RAT tool.

15.484.3 Member Function Documentation

15.484.3.1 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.

15.484.3.2 attributeHash() `std::size_t benchmax::Tool::attributeHash ( ) const [inline], [inherited]`

Hash of the attributes.

15.484.3.3 attributes() `const std::map<std::string,std::string>& benchmax::Tool::attributes ( ) const [inline], [inherited]`

A set of attributes, for example compilation options.

15.484.3.4 binary() `fs::path benchmax::Tool::binary ( ) const [inline], [inherited]`

Full path to the binary.

15.484.3.5 canHandle() `virtual bool benchmax::SMTRAT_Analyzer::canHandle ( const fs::path & path ) const [inline], [override], [virtual]`

Only handle .smt2 files.

Reimplemented from benchmax::Tool.

15.484.3.6 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.

15.484.3.7 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.

15.484.3.8 name() `std::string benchmax::Tool::name ( ) const [inline], [inherited]`

Common name of this tool.

15.484.3.9 parseCommandline() `virtual std::optional<std::string> benchmax::Tool::parse↩ Commandline ( const std::string & cmdline ) const [inline], [virtual], [inherited]`

Compose commandline for this tool and the given input file.

15.484.3.10 resolveDependencies() `std::vector<std::string> benchmax::Tool::resolveDependencies ( ) const [inline], [inherited]`

Get dependencies of binary required to run it (via ldd)

15.484.4 Field Documentation

15.484.4.1 mArguments `std::string benchmax::Tool::mArguments [protected], [inherited]`

Command line arguments that should be passed to the binary.

15.484.4.2 mAttributes `std::map<std::string,std::string> benchmax::Tool::mAttributes [protected], [inherited]`

Attributes of the tool obtained by introspection of the binary.

15.484.4.3 mBinary `fs::path benchmax::Tool::mBinary [protected], [inherited]`

Path to the binary.

15.484.4.4 mName `std::string benchmax::Tool::mName [protected], [inherited]`

(Non-unique) identifier for the tool.

15.485 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.

15.485.1 Detailed Description

Adapts the SMTRAT solver for OPB files.

15.485.2 Constructor & Destructor Documentation

15.485.2.1 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.

15.485.3 Member Function Documentation

15.485.3.1 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.

15.485.3.2 attributeHash()

```
std::size_t benchmax::Tool::attributeHash ( ) const [inline], [inherited]
```

Hash of the attributes.

15.485.3.3 attributes()

```
const std::map<std::string,std::string>& benchmax::Tool::attributes ( )
const [inline], [inherited]
```

A set of attributes, for example compilation options.

15.485.3.4 binary()

```
fs::path benchmax::Tool::binary ( ) const [inline], [inherited]
```

Full path to the binary.

15.485.3.5 canHandle()

```
virtual bool benchmax::SMTRAT_OPB::canHandle (
    const fs::path & path ) const [inline], [override], [virtual]
```

Only handles .opb files.

Reimplemented from benchmax::SMTRAT.

15.485.3.6 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.

15.485.3.7 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.

15.485.3.8 getStatusFromOutput()

```
std::string benchmax::SMTRAT::getStatusFromOutput (
    const BenchmarkResult & result ) const [inline], [inherited]
```

Try to parse memouts from stderr.

15.485.3.9 name()

```
std::string benchmax::Tool::name ( ) const [inline], [inherited]
```

Common name of this tool.

15.485.3.10 parse_stats()

```
bool benchmax::SMTRAT::parse_stats (
    BenchmarkResult & result ) const [inline], [inherited]
```

15.485.3.11 parseCommandline()

```
virtual std::optional<std::string> benchmax::Tool::parse↩
Commandline (
    const std::string & cmdline ) const [inline], [virtual], [inherited]
```

Compose commandline for this tool and the given input file.

15.485.3.12 resolveDependencies()

```
std::vector<std::string> benchmax::Tool::resolveDependencies
( ) const [inline], [inherited]
```

Get dependencies of binary required to run it (via ldd)

15.485.4 Field Documentation

15.485.4.1 mArguments

```
std::string benchmax::Tool::mArguments [protected], [inherited]
```

Command line arguments that should be passed to the binary.

15.485.4.2 mAttributes

```
std::map<std::string,std::string> benchmax::Tool::mAttributes [protected],
[inherited]
```

Attributes of the tool obtained by introspection of the binary.

15.485.4.3 mBinary `fs::path benchmax::Tool::mBinary [protected], [inherited]`

Path to the binary.

15.485.4.4 mName `std::string benchmax::Tool::mName [protected], [inherited]`

(Non-unique) identifier for the tool.

15.486 smtrat::execution::SoftAssertion Struct Reference

`#include <ExecutionState.h>`

Data Fields

- FormulaT formula
- Rational weight
- std::string id

15.486.1 Field Documentation

15.486.1.1 formula `FormulaT smtrat::execution::SoftAssertion::formula`

15.486.1.2 id `std::string smtrat::execution::SoftAssertion::id`

15.486.1.3 weight `Rational smtrat::execution::SoftAssertion::weight`

15.487 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

15.487.1 Field Documentation

15.487.1.1 convert_to_cnf_dimacs `bool smtrat::settings::SolverSettings::convert_to_cnf_dimacs`

15.487.1.2 convert_to_cnf_smtlib `bool smtrat::settings::SolverSettings::convert_to_cnf_smtlib`

15.487.1.3 preprocess `bool smtrat::settings::SolverSettings::preprocess`

15.487.1.4 preprocess_output_file `std::string smtrat::settings::SolverSettings::preprocess_output_file`

15.487.1.5 print_all_models `bool smtrat::settings::SolverSettings::print_all_models`

15.487.1.6 print_model `bool smtrat::settings::SolverSettings::print_model`

15.488 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

15.488.1 Constructor & Destructor Documentation

15.488.1.1 SortedVariableParser() `smtrat::parser::SortedVariableParser::SortedVariableParser ( ) [inline]`

15.488.2 Field Documentation

15.488.2.1 main `qi::rule<Iterator, std::pair<std::string, carl::Sort>), Skipper> smtrat::parser::SortedVariableParser::main`

15.488.2.2 sort `SortParser smtrat::parser::SortedVariableParser::sort`

15.488.2.3 symbol `SymbolParser smtrat::parser::SortedVariableParser::symbol`

15.489 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)

15.489.1 Constructor & Destructor Documentation

15.489.1.1 SortParser() `smtrat::parser::SortParser::SortParser ( ) [inline]`

15.489.2 Member Function Documentation

15.489.2.1 clearParameters() `void smtrat::parser::SortParser::clearParameters ( ) [inline]`

15.489.2.2 getSort() `carl::Sort smtrat::parser::SortParser::getSort ( const Identifier & i ) [inline]`

15.489.2.3 getSortWithParam() `carl::Sort smtrat::parser::SortParser::getSortWithParam ( const Identifier & i, const std::vector< carl::Sort > & params ) [inline]`

15.489.2.4 setParameters() `void smtrat::parser::SortParser::setParameters ( const std::vector< std::string > & params ) [inline]`

15.490 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

15.490.1 Constructor & Destructor Documentation

15.490.1.1 SpecConstantParser() `smtrat::parser::SpecConstantParser::SpecConstantParser ( ) [inline]`

15.490.2 Field Documentation

15.490.2.1 binary `BinaryParser smtrat::parser::SpecConstantParser::binary`

15.490.2.2 decimal `DecimalParser smtrat::parser::SpecConstantParser::decimal`

15.490.2.3 hexadecimal `HexadecimalParser smtrat::parser::SpecConstantParser::hexadecimal`

15.490.2.4 main `qi::rule<Iterator, types::ConstType(), Skipper> smtrat::parser::SpecConstant↩ Parser::main`

15.490.2.5 numeral `NumeralParser smtrat::parser::SpecConstantParser::numeral`

15.490.2.6 string `StringParser smtrat::parser::SpecConstantParser::string`

15.490.2.7 theoryConst `qi::symbols<char, types::ConstType> smtrat::parser::SpecConstant↩ Parser::theoryConst`

15.491 smtrat::SplitSOSModule< Settings > Class Template Reference

`#include <SplitSOSModule.h>`

Public Types

- typedef Settings SettingsType
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::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::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
- 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 _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.
- 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 &_branching↩Value) 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 _preferLeft↩Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩::vector< FormulaT >())

- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩CaseWeak=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.

15.491.1 Member Typedef Documentation

15.491.1.1 SettingsType

```
template<typename Settings >
typedef Settings smtrat::SplitSOSModule< Settings >::SettingsType
```

15.491.2 Member Enumeration Documentation

15.491.2.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

15.491.3 Constructor & Destructor Documentation

15.491.3.1 SplitSOSModule()

```
template<class Settings >
smtrat::SplitSOSModule< Settings >::SplitSOSModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = NULL )
```

15.491.3.2 ~SplitSOSModule()

```
template<class Settings >
smtrat::SplitSOSModule< Settings >::~SplitSOSModule ( )
```

15.491.4 Member Function Documentation

15.491.4.1 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

_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.

15.491.4.2 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 >.

15.491.4.3 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

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 smtrat::SATModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::VSModule< Settings >, smtrat::PFEModule< Settings >, smtrat::EQModule< Setting smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Sett smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, and smtrat::GBModule< Settings >.

15.491.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.491.4.5 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.

15.491.4.6 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.

15.491.4.7 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.

15.491.4.8 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.

15.491.4.9 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.

15.491.4.10 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

_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.

15.491.4.11 allModels() `const std::vector<Model>& smtrat::Module::allModels ( ) const [inline], [inherited]`

Returns All satisfying assignments, if existent.

15.491.4.12 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.

15.491.4.13 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.

15.491.4.14 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.

15.491.4.15 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.

15.491.4.16 branchAt() [1/6]

```
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]
```

15.491.4.17 branchAt() [2/6]

```
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]
```

15.491.4.18 branchAt() [3/6]

```
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]
```

15.491.4.19 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.491.4.20 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P
>> = carl::dummy>
bool smtrat::Module::branchAt (
    const typename P::PolyType & _poly,
```

```
            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]
```

15.491.4.21 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P
>> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false )  [inline], [protected], [inherited]
```

15.491.4.22 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

15.491.4.23 checkCore()

```
template<class Settings >
Answer smtrat::SplitSOSModule< 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.

15.491.4.24 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
    std::vector< FormulaSetT >::const_iterator _infsubset,
    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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.491.4.25 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.

15.491.4.26 cleanModel()

```
void smtrat::Module::cleanModel ( ) const [inline], [protected],
[inherited]
```

Substitutes variable occurrences by its model value in the model values of other variables.

15.491.4.27 clearLemmas()

```
void smtrat::Module::clearLemmas ( ) [inline], [inherited]
```

Deletes all yet found lemmas.

15.491.4.28 clearModel()

```
void smtrat::Module::clearModel ( ) const [inline], [protected],
[inherited]
```

Clears the assignment, if any was found.

15.491.4.29 clearModels()

```
void smtrat::Module::clearModels ( ) const [inline], [protected],
[inherited]
```

Clears all assignments, if any was found.

15.491.4.30 clearPassedFormula()

```
void smtrat::Module::clearPassedFormula ( ) [protected],
[inherited]
```

15.491.4.31 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

15.491.4.32 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.

15.491.4.33 collectTheoryPropagations()

```
void smtrat::Module::collectTheoryPropagations ( ) [inherited]
```

15.491.4.34 constraintsToInform()

```
const carl::FastSet<FormulaT>& smtrat::Module::constraints↩
ToInform ( ) const [inline], [inherited]
```

Returns The constraints which the backends must be informed about.

15.491.4.35 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

15.491.4.36 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)

15.491.4.37 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

_constraint	The constraint to remove from internal data structures.

15.491.4.38 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.491.4.39 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.491.4.40 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.491.4.41 excludeNotReceivedVariablesFromModel()

`void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

15.491.4.42 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.491.4.43 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.

15.491.4.44 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.

15.491.4.45 freeSplittingVariable()

```
static void smtrat::Module::freeSplittingVariable (
    const FormulaT & _splittingVariable ) [inline], [static], [inherited]
```

15.491.4.46 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

15.491.4.47 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

15.491.4.48 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

15.491.4.49 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

15.491.4.50 getInfeasibleSubsets() [2/2] `std::vector< FormulaSetT > smtrat::Module::getInfeasible↩ Subsets ( 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.491.4.51 getInformationRelevantFormulas() `const std::set< FormulaT > & smtrat::Module::get↩ InformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.491.4.52 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.

15.491.4.53 getOrigins() [1/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

15.491.4.54 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.491.4.55 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.491.4.56 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 reveived formula, if the Boolean is true.

Reimplemented from smtrat::Module.

15.491.4.57 hasLemmas()

```
bool smtrat::Module::hasLemmas ( ) [inline], [inherited]
```

Checks whether this module or any of its backends provides any lemmas.

15.491.4.58 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).

15.491.4.59 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns A unique ID to identify this module instance.

15.491.4.60 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.

15.491.4.61 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.491.4.62 informBackends() `void smtrat::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.

15.491.4.63 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, and smtrat::ICPModule< ICPSettings1 >.

15.491.4.64 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.491.4.65 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::FPPModule< Settings >.

15.491.4.66 is_minimizing()

```
bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]
```

15.491.4.67 isLemmaLevel()

```
bool smtrat::Module::isLemmaLevel (
    LemmaLevel level ) [protected], [inherited]
```

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.491.4.68 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().

15.491.4.69 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.

15.491.4.70 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.491.4.71 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.491.4.72 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.

15.491.4.73 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.

15.491.4.74 objective()

```
carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]
```

15.491.4.75 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
    const FormulaT & _origin ) const [protected], [inherited]
```

15.491.4.76 passedFormulaBegin()

```
ModuleInput::iterator smtrat::Module::passedFormulaBegin ( )
[inline], [protected], [inherited]
```

Returns An iterator to the end of the passed formula. TODO: disable this method

15.491.4.77 passedFormulaEnd()

```
ModuleInput::iterator smtrat::Module::passedFormulaEnd ( )
[inline], [protected], [inherited]
```

Returns An iterator to the end of the passed formula.

15.491.4.78 pPassedFormula()

```
const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline],
[inherited]
```

Returns A pointer to the formula passed to the backends of this module.

15.491.4.79 pReceivedFormula()

```
const ModuleInput* smtrat::Module::pReceivedFormula ( ) const
[inline], [inherited]
```

Returns A pointer to the formula passed to this module.

15.491.4.80 print()

```
void smtrat::Module::print (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.491.4.81 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

_out	The stream to print the assignment on.

15.491.4.82 printInfeasibleSubsets()

`void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.491.4.83 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

_out	The stream to print the assignment on.

15.491.4.84 printPassedFormula()

`void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.491.4.85 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.491.4.86 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.

15.491.4.87 receivedFormulaChecked()

```
void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]
```

Notifies that the received formulas has been checked.

15.491.4.88 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩InfeasibleSubset (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

15.491.4.89 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const [inline], [inherited]
```

15.491.4.90 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

_subformula	The sub formula of the received formula to remove.

Reimplemented from smtrat::Module.

15.491.4.91 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

formula	The sub formula of the received formula to remove.

Reimplemented in smtrat::SATModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::VSModule< Settings >, smtrat::PFEModule< Settings >, smtrat::EQModule< Setting smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Sett smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, and smtrat::GBModule< Settings >.

15.491.4.92 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin
(
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.491.4.93 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

15.491.4.94 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline],
[inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.491.4.95 rReceivedFormula() `const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]`

Returns A reference to the formula passed to this module.

15.491.4.96 runBackends() [1/2] `virtual Answer smtrat::PModule::runBackends ( ) [inline], [virtual], [inherited]`

Reimplemented from smtrat::Module.

15.491.4.97 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.

15.491.4.98 setId()

```
void smtrat::Module::setId (
    std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.491.4.99 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.

15.491.4.100 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.

15.491.4.101 splitUnequalConstraint() `void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.491.4.102 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.491.4.103 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 >.

15.491.4.104 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.491.4.105 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.

15.491.4.106 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).

15.491.5 Field Documentation

15.491.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.491.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.491.5.3 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.

15.491.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.491.5.5 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

15.491.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.491.5.7 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.

15.491.5.8 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.

15.491.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.491.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.491.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.491.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.491.5.13 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.

15.491.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.491.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.491.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.491.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.491.5.18 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.

15.491.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

15.491.5.20 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.491.5.21 mSmallerMusesCheckCounter

`unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.491.5.22 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.

15.491.5.23 mTheoryPropagations

`std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

15.491.5.24 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).

15.491.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.492 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.

15.492.1 Constructor & Destructor Documentation

15.492.1.1 SSHBackend() `benchmax::SSHBackend::SSHBackend ( ) [inline]`

15.492.1.2 ~SSHBackend() `benchmax::SSHBackend::~SSHBackend ( ) [inline]`

15.492.2 Member Function Documentation

15.492.2.1 addResult()

```
void benchmax::Backend::addResult (
    const Tool * tool,
    const fs::path & file,
    BenchmarkResult && result ) [inline], [inherited]
```

Add a result.

15.492.2.2 collect_results()

```
virtual bool benchmax::Backend::collect_results (
    const Jobs & ,
    bool ) [inline], [protected], [virtual], [inherited]
```

15.492.2.3 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.

15.492.2.4 finalize()

```
virtual void benchmax::Backend::finalize ( ) [inline], [protected], [virtual],
[inherited]
```

Hook to allow for asynchronous backends to wait for jobs to terminate.

15.492.2.5 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.

15.492.2.6 process_results()

```
void benchmax::Backend::process_results (
    const Jobs & jobs,
    bool check_finished ) [inline], [inherited]
```

15.492.2.7 run()

```
void benchmax::SSHBackend::run (
    const Jobs & ,
    bool ) [inline]
```

Dummy if SSH is disabled.

15.492.2.8 sanitize_results()

```
void benchmax::Backend::sanitize_results (
    const Jobs & jobs ) const [inline], [protected], [inherited]
```

15.492.2.9 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.

15.492.2.10 suspendable() `bool benchmax::Backend::suspendable ( ) const [inline], [inherited]`

15.492.2.11 write_results() `void benchmax::Backend::write_results ( const Jobs & jobs ) const [inline], [protected], [inherited]`

15.492.3 Field Documentation

15.492.3.1 mExpectedJobs `std::size_t benchmax::Backend::mExpectedJobs [protected], [inherited]`

Number of jobs that should be run.

15.492.3.2 mFinishedJobs `std::atomic<std::size_t> benchmax::Backend::mFinishedJobs [protected], [inherited]`

Number of jobs that are finished.

15.492.3.3 mLastPercent `std::atomic<std::size_t> benchmax::Backend::mLastPercent [protected], [inherited]`

Percentage of finished jobs when madeProgress() was last called.

15.493 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.

15.493.1 Detailed Description

Settings for SSH backend.

15.493.2 Field Documentation

15.493.2.1 basedir `std::string benchmax::settings::SSHBackendSettings::basedir`

Base directory for solvers.

15.493.2.2 nodes `std::vector<std::string> benchmax::settings::SSHBackendSettings::nodes`

List of nodes to connect to.

15.493.2.3 resolve_deps `bool benchmax::settings::SSHBackendSettings::resolve_deps`

Resolve and upload dependencies of binary.

15.493.2.4 tmpdir `std::string benchmax::settings::SSHBackendSettings::tmpdir`

Temporary directory for benchmarks and output files.

15.493.2.5 use_wallclock `bool benchmax::settings::SSHBackendSettings::use_wallclock`

Use wallclock timeouts instead of CPU time.

15.494 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_IRU↩SR|S_IWUSR)

  Upload a file to the remote.
- bool executeCommand (const std::string &cmd, BenchmarkResult &result)

  Execute a command on the remote.

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

15.494.2 Constructor & Destructor Documentation

15.494.2.1 SSHConnection()

```
benchmax::ssh::SSHConnection::SSHConnection (
    const Node & n ) [inline]
```

Create a new connection for the given node.

15.494.2.2 ~SSHConnection()

```
benchmax::ssh::SSHConnection::~SSHConnection ( ) [inline]
```

Wait for all channels to terminate.

15.494.3 Member Function Documentation

15.494.3.1 busy()

```
bool benchmax::ssh::SSHConnection::busy ( ) [inline]
```

Check if all channels are busy.

15.494.3.2 createTmpDir()

```
std::string benchmax::ssh::SSHConnection::createTmpDir (
    const std::string & folder ) [inline]
```

Create a temporary directory on the remote.

15.494.3.3 executeCommand()

```
bool benchmax::ssh::SSHConnection::executeCommand (
    const std::string & cmd,
    BenchmarkResult & result ) [inline]
```

Execute a command on the remote.

15.494.3.4 finishJob()

```
void benchmax::ssh::SSHConnection::finishJob ( ) [inline]
```

Decrease job counter.

15.494.3.5 getNode()

```
const Node& benchmax::ssh::SSHConnection::getNode ( ) const [inline]
```

Return the node.

15.494.3.6 jobFree()

```
bool benchmax::ssh::SSHConnection::jobFree ( ) [inline]
```

Check if a new job could be started.

15.494.3.7 jobs() `std::size_t benchmax::ssh::SSHConnection::jobs ( ) const [inline]`

Current number of jobs.

15.494.3.8 newJob() `void benchmax::ssh::SSHConnection::newJob ( ) [inline]`

Increase job counter.

15.494.3.9 removeDir() `void benchmax::ssh::SSHConnection::removeDir ( const std::string & folder ) [inline]`

Remove a (temporary) directory on the remote.

15.494.3.10 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.

15.495 smtrat::vs::State Class Reference

`#include <State.h>`

Public Types

- enum Type { TEST_CANDIDATE_TO_GENERATE, SUBSTITUTION_TO_APPLY, COMBINE_SUBRESULTS }
- typedef carl::FastPointerMapB< Substitution, ConditionSetSetSet > ConflictSets
- typedef std::vector< std::pair< ConditionList, bool > > SubstitutionResult
- typedef std::vector< SubstitutionResult > SubstitutionResults
- typedef std::vector< std::pair< unsigned, unsigned > > SubResultCombination
- typedef smtrat::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 _with↩ VariableBounds)

  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 smtrat::Rational & minIntTestCandidate () const
- const smtrat::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 smtrat::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 (smtrat::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 smtrat::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 * > &_reactivated↩ States)

  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 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.

- 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 smtrat::ConstraintT &_constraint, const carl::PointerSet< Condition > &_original↩ Conditions, size_t _valutation, 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 _originsAre↩ CurrentConditions)

  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 _include↩ InconsistentTestCandidates=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< smtrat::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="***", 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 ConditionSetSetSet &_conflictSets, carl::PointerSet< Condition > &_minCovSet, unsigned _currentTreeDepth)

  Finds a covering set of a vector of sets of sets due to some heuristics.

15.495.1 Member Typedef Documentation

15.495.1.1 ConflictSets `typedef carl::FastPointerMapB<Substitution,ConditionSetSetSet> smtrat::vs::State::Conf`

15.495.1.2 SubResultCombination `typedef std::vector<std::pair< unsigned, unsigned> > smtrat::vs::State::Sub`

15.495.1.3 SubstitutionResult `typedef std::vector<std::pair< ConditionList, bool> > smtrat::vs::State::Substi`

15.495.1.4 SubstitutionResults `typedef std::vector<SubstitutionResult> smtrat::vs::State::SubstitutionResults`

15.495.1.5 VariableBoundsCond `typedef smtrat::vb::VariableBounds<const Condition*> smtrat::vs::State::Variab`

15.495.2 Member Enumeration Documentation

15.495.2.1 Type `enum smtrat::vs::State::Type`

Enumerator

TEST_CANDIDATE_TO_GENERATE	
SUBSTITUTION_TO_APPLY	
COMBINE_SUBRESULTS	

15.495.3 Constructor & Destructor Documentation

15.495.3.1 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

_withVariableBounds	A flag that indicates whether to use optimizations based on variable bounds.

15.495.3.2 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

_father	The father of the state to be constructed.
_substitution	The substitution of the state to be constructed.
_withVariableBounds	A flag that indicates whether to use optimizations based on variable bounds.

15.495.3.3 State() [3/3]

```
smtrat::vs::State::State (
    const State & ) [delete]
```

15.495.3.4 ~State()

```
smtrat::vs::State::~State ( )
```

Destructor.

15.495.4 Member Function Documentation

15.495.4.1 addChild()

```
std::vector< State * > smtrat::vs::State::addChild (
    const Substitution & _substitution )
```

Adds a state as child to this state with the given substitution.

Parameters

_substitution	The substitution to generate the state for.

Returns true, if a state has been added as child to this state; false, otherwise.

15.495.4.2 addCondition()

```
void smtrat::vs::State::addCondition (
    const smtrat::ConstraintT & _constraint,
    const carl::PointerSet< Condition > & _originalConditions,
    size_t _valutation,
    bool _recentlyAdded,
    ValuationMap & _ranking )
```

Adds a constraint to the conditions of this state.

Parameters

_constraint	The constraint of the condition to add.
_originalConditions	The original conditions of the condition to add.
_valutation	The valuation of the condition to add.
_recentlyAdded	Is the condition a recently added one. @sideeffect The state can obtain a new condition.

15.495.4.3 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

_substitution	The corresponding substitution generated the conflict. (NULL in the case a detected conflict without substitution)
_condSetSet	The conflicts to add.

15.495.4.4 addConflictSet()

```
void smtrat::vs::State::addConflictSet (
    const Substitution * _substitution,
    ConditionSetSet && _condSetSet )
```

Adds a conflict set to the map of substitutions to conflict sets.

Parameters

_substitution	The corresponding substitution generated the conflict. (NULL in the case a detected conflict without substitution)
_condSetSet	The conflicts to add.

15.495.4.5 addSubstitutionResults()

```
void smtrat::vs::State::addSubstitutionResults (
    std::vector< DisjunctionOfConditionConjunctions > && _disjunctionsOfCondConj )
```

Adds the given substitution results to this state.

Parameters

_disjunctionsOfCondConj	The substitution results given by a vector of disjunctions of conjunctions of conditions.

15.495.4.6 allTestCandidatesInvalidated()

```
bool smtrat::vs::State::allTestCandidatesInvalidated (
    const Condition * _condition ) const
```

Parameters

A	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.

15.495.4.7 averageConstraintValuation()

```
void smtrat::vs::State::averageConstraintValuation (
    const std::vector< std::pair< carl::Variable, std::multiset< double >>> & _varVals )
```

15.495.4.8 backendCallValuation()

```
size_t smtrat::vs::State::backendCallValuation ( ) const [inline]
```

Returns The valuation of the state's considered conditions for a possible backend call.

15.495.4.9 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

_bestCondition	The most adequate condition to be the next test candidate provider.
_numberOfAllVariables	The number of all globally known variables.
_preferEquation	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.

15.495.4.10 bestConstraintValuation()

```
void smtrat::vs::State::bestConstraintValuation (
    const std::vector< std::pair< carl::Variable, std::multiset< double >>> & _varVals )
```

15.495.4.11 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.

15.495.4.12 checkSubresultCombinations()

```
bool smtrat::vs::State::checkSubresultCombinations ( ) const
```

This is just for debug purpose.

Returns true, if no subresult combination is illegal.

15.495.4.13 checkTestCandidatesForBounds()

```
bool smtrat::vs::State::checkTestCandidatesForBounds ( )
```

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.

15.495.4.14 children()

```
const std::list<State*>& smtrat::vs::State::children ( ) const [inline]
```

Returns A constant reference to the vector of the children of this state.

15.495.4.15 conditions()

```
const ConditionList& smtrat::vs::State::conditions ( ) const [inline]
```

Returns A constant reference to the currently considered conditions of this state.

15.495.4.16 conditionsSimplified()

```
bool smtrat::vs::State::conditionsSimplified ( ) const [inline]
```

Returns true, if the considered conditions of this state are already simplified.

15.495.4.17 conflictSets()

```
const ConflictSets& smtrat::vs::State::conflictSets ( ) const [inline]
```

Returns A constant reference to the conflict sets of this state.

15.495.4.18 constraintExists()

```
ConditionList::iterator smtrat::vs::State::constraintExists (
    const smtrat::ConstraintT & _constraint )
```

Checks if the given constraint already exists as condition in the state.

Parameters

_constraint	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.

15.495.4.19 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.

15.495.4.20 coveringSet()

```
size_t smtrat::vs::State::coveringSet (
    const ConditionSetSetSet & _conflictSets,
    carl::PointerSet< Condition > & _minCovSet,
    unsigned _currentTreeDepth )  [static]
```

Finds a covering set of a vector of sets of sets due to some heuristics.

Parameters

_conflictSets	The vector of sets of sets, for which the method finds all minimum covering sets.
_minCovSet	The found mininum covering set.
_currentTreeDepth	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.

15.495.4.21 currentRangeSize()

```
size_t smtrat::vs::State::currentRangeSize ( ) const  [inline]
```

15.495.4.22 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

_conditionsToDelete	The conditions to delete.

15.495.4.23 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

_originsToDelete	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.

15.495.4.24 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

_originsToDelete	The condition for which to delete everything originated by them.

15.495.4.25 deleteOriginsFromConflictSets()

```
void smtrat::vs::State::deleteOriginsFromConflict↩
Sets (
    carl::PointerSet< Condition > & _originsToDelete,
    bool _originsAreCurrentConditions )
```

Deletes everything originated by the given conditions in the conflict sets of this state.

Parameters

_originsToDelete	The condition for which to delete everything originated by them.
_originsAreCurrentConditions	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.

15.495.4.26 deleteOriginsFromSubstitutionResults()

```
void smtrat::vs::State::deleteOriginsFrom↩
SubstitutionResults (
    carl::PointerSet< Condition > & _originsToDelete )
```

Deletes everything originated by the given conditions in the substitution results of this state.

Parameters

_originsToDelete	The conditions for which to delete everything originated by them.

15.495.4.27 extendSubResultCombination() `bool smtrat::vs::State::extendSubResultCombination ( )`

Extends the currently considered combination of conjunctions in the substitution results.

15.495.4.28 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.

15.495.4.29 getCurrentSubresultCombination() `ConditionList smtrat::vs::State::getCurrent↩SubresultCombination ( ) const`

Gets the current substitution result combination as condition vector.

Returns The current substitution result combination as condition vector.

15.495.4.30 getNextIntTestCandidate() `bool smtrat::vs::State::getNextIntTestCandidate ( smtrat::Rational & _nextIntTestCandidate, size_t _maxIntRange )`

15.495.4.31 getNumberOfCurrentSubresultCombination() `size_t smtrat::vs::State::getNumberOf↩CurrentSubresultCombination ( ) const`

Returns The number of the current substitution result combination.

15.495.4.32 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.

15.495.4.33 hasChildWithID() `bool smtrat::vs::State::hasChildWithID ( ) const`

Checks whether a child exists, which has no ID (!=0).

Returns true, if there exists a child with ID (!=0); false, otherwise.

15.495.4.34 hasFurtherUncheckedTestCandidates() `bool smtrat::vs::State::hasFurtherUnchecked↩ TestCandidates ( ) 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.

15.495.4.35 hasInfinityChild() `bool smtrat::vs::State::hasInfinityChild ( ) const [inline]`

15.495.4.36 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.

15.495.4.37 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.

15.495.4.38 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.

15.495.4.39 hasRecentlyAddedConditions() `bool smtrat::vs::State::hasRecentlyAddedConditions ( ) const [inline]`

Returns true, if this state has a recently added condition in its considered conditions.

15.495.4.40 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

_condition	The condition to check.
A	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.

15.495.4.41 hasSubResultsCombination() `bool smtrat::vs::State::hasSubResultsCombination ( ) const [inline]`

Returns true, if this state has a substitution result combination.

15.495.4.42 hasSubstitutionResults() `bool smtrat::vs::State::hasSubstitutionResults ( ) const [inline]`

Returns true, if this state has substitution results.

15.495.4.43 id() `size_t smtrat::vs::State::id ( ) const [inline]`

Returns The id of this state.

15.495.4.44 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.

15.495.4.45 initConditionFlags() `void smtrat::vs::State::initConditionFlags ( )`

Sets all flags of the conditions to true, if it contains the variable given by the states index.

15.495.4.46 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

_allVariables	All globally known variables.
_vvstrat	The strategy according which we choose the variable.
_preferEquation	A flag that indicates to prefer equations in the heuristics of this method.
_tryDifferentVarOrder	

15.495.4.47 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.

15.495.4.48 isRoot() `bool smtrat::vs::State::isRoot ( ) const [inline]`

Returns The root of the state tree where this state is part from.

15.495.4.49 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.

15.495.4.50 maxIntTestCandidate() `const smtrat::Rational& smtrat::vs::State::maxIntTestCandidate ( ) const [inline]`

15.495.4.51 minIntTestCandidate() `const smtrat::Rational& smtrat::vs::State::minIntTestCandidate ( ) const [inline]`

15.495.4.52 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.

15.495.4.53 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.

15.495.4.54 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.

15.495.4.55 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.

15.495.4.56 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

_checkConflictForSideCondition	If this flag is set to false it might save some computations but leads to an over-approximative behavior. Recommendation: set it to true.
_includeInconsistentTestCandidates	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.
_useBackjumping	If true, we use the backjumping technique.

15.495.4.57 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.

15.495.4.58 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.

15.495.4.59 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

_initiation	A string which is printed in the beginning of every row.
_out	The stream to print on.

15.495.4.60 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

_initiation	A string which is printed in the beginning of every row.
_out	The stream to print on.

15.495.4.61 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

_initiation	A string which is printed in the beginning of every row.
_out	The stream to print on.

15.495.4.62 printConflictSets()

```
void smtrat::vs::State::printConflictSets (
    const std::string _initiation = "***",
    std::ostream & _out = std::cout ) const
```

Prints the conflict sets of this state.

Parameters

_initiation	A string which is printed in the beginning of every row.
_out	The stream to print on.

15.495.4.63 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

_initiation	A string which is printed in the beginning of every row.
_out	The stream to print on.

15.495.4.64 printSubstitutionResultCombinationAsNumbers()

```
void smtrat::vs::State::printSubstitution↩
ResultCombinationAsNumbers (
            const std::string _initiation = "***",
            std::ostream & _out = std::cout ) const
```

Prints the combination of substitution results, expressed in numbers, used in this state.

Parameters

_initiation	A string which is printed in the beginning of every row.
_out	The stream to print on.

15.495.4.65 printSubstitutionResults()

```
void smtrat::vs::State::printSubstitutionResults (
            const std::string _initiation = "***",
            std::ostream & _out = std::cout ) const
```

Prints the substitution results of this state.

Parameters

_initiation	A string which is printed in the beginning of every row.
_out	The stream to print on.

15.495.4.66 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.

15.495.4.67 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.

15.495.4.68 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.

15.495.4.69 rChildren() `std::list<State*>& smtrat::vs::State::rChildren ( ) [inline]`

Returns A reference to the vector of the children of this state.

15.495.4.70 rConditions() `ConditionList& smtrat::vs::State::rConditions ( ) [inline]`

Returns A reference to the currently considered conditions of this state.

15.495.4.71 rConflictSets() `ConflictSets& smtrat::vs::State::rConflictSets ( ) [inline]`

Returns A reference to the conflict sets of this state.

15.495.4.72 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.

15.495.4.73 removeStatesFromRanking() `void smtrat::vs::State::removeStatesFromRanking ( const State & toRemove, ValuationMap & _ranking ) [static]`

15.495.4.74 resetCannotBeSolvedLazyFlags() `void smtrat::vs::State::resetCannotBeSolvedLazy↩ Flags ( )`

15.495.4.75 resetConflictSets() `void smtrat::vs::State::resetConflictSets ( )`

Clears the conflict sets.

15.495.4.76 resetCurrentRangeSize() `void smtrat::vs::State::resetCurrentRangeSize ( ) [inline]`

15.495.4.77 resetInfinityChild() `void smtrat::vs::State::resetInfinityChild ( const State * _state ) [inline]`

15.495.4.78 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.

15.495.4.79 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.

15.495.4.80 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.

15.495.4.81 rID() `size_t& smtrat::vs::State::rID ( ) [inline]`

Returns A reference to the id of this state.

15.495.4.82 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.

15.495.4.83 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.

15.495.4.84 root() `State & smtrat::vs::State::root ( )`

Returns The root of the tree, in which this state is located.

15.495.4.85 rSubResultCombination() `SubResultCombination& smtrat::vs::State::rSubResult↩ Combination ( ) [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())

15.495.4.86 rSubResultsSimplified() `bool& smtrat::vs::State::rSubResultsSimplified ( ) [inline]`

Returns A reference to the flag indicating whether the substitution results of this state are already simplified.

15.495.4.87 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.

15.495.4.88 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())

15.495.4.89 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.

15.495.4.90 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.

15.495.4.91 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.

15.495.4.92 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.

15.495.4.93 setIndex() `void smtrat::vs::State::setIndex ( const carl::Variable & _index )`

Sets the index of this state.

Parameters

_index	The string to which the index should be set.

15.495.4.94 setInfinityChild() `void smtrat::vs::State::setInfinityChild ( State * _state ) [inline]`

15.495.4.95 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

_pOCondition	The pointer to set the original condition to.

15.495.4.96 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

_conditionVectorToSimplify	The conditions to simplify. Note, that this method can change these conditions.
_conflictSet	All conflicts this method detects in the given conditions are stored in here.
_stateConditions	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.

15.495.4.97 simplify() [2/2]

```
void smtrat::vs::State::simplify (
    ValuationMap & _ranking )
```

Cleans up all conditions in this state according to comparison between the corresponding constraints.

15.495.4.98 solutionSpace()

```
std::vector< smtrat::DoubleInterval > smtrat::vs::State::solution↩
Space (
    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

_conflictReason	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.

15.495.4.99 subResultCombination()

```
const SubResultCombination& smtrat::vs::State::subResult↩
Combination ( ) 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())

15.495.4.100 subResultsSimplified() `bool smtrat::vs::State::subResultsSimplified ( ) const [inline]`

Returns true, if the substitution results of this state are already simplified.

15.495.4.101 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.

15.495.4.102 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.

15.495.4.103 substitutionApplicable() [2/2] `bool smtrat::vs::State::substitutionApplicable ( const smtrat::ConstraintT & _constraint ) const`

Checks if the substitution of this state can be applied to the given constraint.

Parameters

_constraint	The constraint, for which we want to know, if the substitution is applicable.

Returns true, if the substitution can be applied; false, otherwise.

15.495.4.104 substitutionResults() `const SubstitutionResults& smtrat::vs::State::substitution↩ Results ( ) 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())

15.495.4.105 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.

15.495.4.106 tooHighDegreeConditions()

`const carl::PointerSet<Condition>& smtrat::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.

15.495.4.107 treeDepth()

`unsigned smtrat::vs::State::treeDepth ( ) const`

Returns The depth of the subtree with this state as root node.

15.495.4.108 tryToRefreshIndex()

`bool smtrat::vs::State::tryToRefreshIndex ( ) const [inline]`

Returns true, if the index variable of this state shall be recalculated.

15.495.4.109 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.

15.495.4.110 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.

15.495.4.111 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

_unfinAnt	The unfinished ancestor node.

Returns true, if it has a unfinished ancestor; false, otherwise.

15.495.4.112 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.

15.495.4.113 updateIntTestCandidates() `bool smtrat::vs::State::updateIntTestCandidates ( )`

15.495.4.114 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

_substitution	The substitution containing the test candidate to check for.
_reactivatedStates	

Returns true, if the test candidate of the given substitution was already generated; false, otherwise.

15.495.4.115 updateValuation() `void smtrat::vs::State::updateValuation (`

```
bool _lazy )
```

Updates the valuation of this state.

15.495.4.116 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).

15.495.4.117 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.

15.495.4.118 variables() `void smtrat::vs::State::variables (`

```
carl::Variables & _variables ) const
```

Finds the variables, which occur in this state.

Parameters

_variables	The variables which occur in this state.

15.495.4.119 worstConstraintValuation()

```
void smtrat::vs::State::worstConstraintValuation (
    const std::vector< std::pair< carl::Variable, std::multiset< double >>> & _varVals )
```

15.496 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

15.496.1 Member Typedef Documentation

15.496.1.1 Parents

```
using smtrat::StaticUnionFind::Parents = std::vector<Value>
```

15.496.1.2 Ranks

```
using smtrat::StaticUnionFind::Ranks = std::vector<size_t>
```

15.496.1.3 Representative

```
using smtrat::StaticUnionFind::Representative = Value
```

15.496.1.4 Value

```
using smtrat::StaticUnionFind::Value = size_t
```

15.496.2 Member Function Documentation

15.496.2.1 find()

```
auto smtrat::StaticUnionFind::find (
    Value const & val ) -> Representative   [inline], [noexcept]
```

15.496.2.2 introduce_variable()

```
void smtrat::StaticUnionFind::introduce_variable (
    Value const & var )  [inline], [noexcept]
```

15.496.2.3 merge()

```
void smtrat::StaticUnionFind::merge (
    Value const & a,
    Value const & b )  [inline], [noexcept]
```

15.497 smtrat::statistics::StatisticsSettings Struct Reference

`#include <StatisticsSettings.h>`

Data Fields

- bool export_as_xml
- std::string xml_filename
- bool print_as_smtlib

15.497.1 Field Documentation

15.497.1.1 export_as_xml `bool smtrat::statistics::StatisticsSettings::export_as_xml`

15.497.1.2 print_as_smtlib `bool smtrat::statistics::StatisticsSettings::print_as_smtlib`

15.497.1.3 xml_filename `std::string smtrat::statistics::StatisticsSettings::xml_filename`

15.498 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)

15.498.1 Constructor & Destructor Documentation

15.498.1.1 StrategyGraph() `smtrat::StrategyGraph::StrategyGraph ( ) [inline]`

15.498.2 Member Function Documentation

15.498.2.1 addBackend()

```
template<typename Module >
BackendLink smtrat::StrategyGraph::addBackend (
            const std::initializer_list< BackendLink > & backends ) [inline]
```

15.498.2.2 addEdge()

```
BackendLink& smtrat::StrategyGraph::addEdge (
            std::size_t from,
            std::size_t to ) [inline]
```

15.498.2.3 addRoot()

```
std::size_t smtrat::StrategyGraph::addRoot (
    const std::initializer_list< BackendLink > & backends ) [inline]
```

15.498.2.4 getBackends()

```
std::set<std::pair<thread_priority,AbstractModuleFactory*> > smtrat↩
::StrategyGraph::getBackends (
    std::size_t vertex,
    const carl::Condition & condition ) const [inline]
```

15.498.2.5 getRoot()

```
std::size_t smtrat::StrategyGraph::getRoot ( ) const [inline]
```

15.498.2.6 hasBranches()

```
bool smtrat::StrategyGraph::hasBranches ( ) const [inline]
```

15.498.2.7 numberOfBranches()

```
std::size_t smtrat::StrategyGraph::numberOfBranches ( ) const
[inline]
```

15.498.2.8 TrueCondition()

```
static bool smtrat::StrategyGraph::TrueCondition (
    const carl::Condition & c ) [inline], [static]
```

15.498.3 Friends And Related Function Documentation

15.498.3.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const StrategyGraph & sg ) [friend]
```

15.499 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.

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

15.499.2 Member Function Documentation

15.499.2.1 operator std::string()

```
delta::String::operator std::string ( ) const [inline]
```

Return data as string.

Returns Data as string.

15.499.2.2 operator<<()

```
template<typename T >
String& delta::String::operator<< (
    const T & t ) [inline]
```

Add some data to the string.

Parameters

t	Data.

Returns `*this`

15.500 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

15.500.1 Constructor & Destructor Documentation

15.500.1.1 StringOption()

```
Minisat::StringOption::StringOption (
    const char * c,
    const char * n,
    const char * d,
    const char * def = NULL ) [inline]
```

15.500.2 Member Function Documentation

15.500.2.1 getHelpPrefixString()

`static const char*& Minisat::Option::getHelpPrefixString ( ) [inline], [static], [protected], [inherited]`

15.500.2.2 getOptionList()

`static vec<Option*>& Minisat::Option::getOptionList ( ) [inline], [static], [protected], [inherited]`

15.500.2.3 getUsageString()

`static const char*& Minisat::Option::getUsageString ( ) [inline], [static], [protected], [inherited]`

15.500.2.4 help()

```
virtual void Minisat::StringOption::help (
    bool verbose = false ) [inline], [virtual]
```

Implements Minisat::Option.

15.500.2.5 operator const char *() `Minisat::StringOption::operator const char * ( void ) const [inline]`

15.500.2.6 operator=() `StringOption& Minisat::StringOption::operator= ( const char * x ) [inline]`

15.500.3 Field Documentation

15.500.3.1 category `const char* Minisat::Option::category [protected], [inherited]`

15.500.3.2 description `const char* Minisat::Option::description [protected], [inherited]`

15.500.3.3 name `const char* Minisat::Option::name [protected], [inherited]`

15.500.3.4 type_name `const char* Minisat::Option::type_name [protected], [inherited]`

15.501 smtrat::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

15.501.1 Detailed Description

Parses strings: "."+" with escape sequences \\" and \\\\

15.501.2 Constructor & Destructor Documentation

15.501.2.1 StringParser() `smtrat::parser::StringParser::StringParser ( ) [inline]`

15.501.3 Field Documentation

15.501.3.1 escapes `qi::symbols<char, char> smtrat::parser::StringParser::escapes`

15.501.3.2 main `qi::rule<Iterator, std::string(), Skipper> smtrat::parser::StringParser::main`

15.502 smtrat::STropModule< Settings > Class Template Reference

`#include <STropModule.h>`

Public Types

- typedef Settings SettingsType
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::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 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 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 _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)

  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 &_branching↩ Value) 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 _preferLeft↩ Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩ ::vector< FormulaT >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩ AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩ CaseWeak=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.

15.502.1 Member Typedef Documentation

15.502.1.1 SettingsType `template<typename Settings >`

```
typedef Settings smtrat::STropModule< Settings >::SettingsType
```

15.502.2 Member Enumeration Documentation

15.502.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

15.502.3 Constructor & Destructor Documentation

15.502.3.1 STropModule()

```
template<class Settings >
smtrat::STropModule< Settings >::STropModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

15.502.4 Member Function Documentation

15.502.4.1 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.

15.502.4.2 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 >.

15.502.4.3 addCore()

```
template<class 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

_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.

Normalize the left hand side of the constraint and turn the relation accordingly

Store the normalized constraint and mark the separator object as changed

Check if the asserted constraint prohibits the application of this method

Check if the asserted relation trivially conflicts with other asserted relations

Reimplemented from smtrat::Module.

15.502.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.502.4.5 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.

15.502.4.6 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.

15.502.4.7 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.

15.502.4.8 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.

15.502.4.9 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.

15.502.4.10 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

_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.

15.502.4.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns All satisfying assignments, if existent.

15.502.4.12 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.

15.502.4.13 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.

15.502.4.14 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.

15.502.4.15 branchAt() [1/6]

```
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]
```

15.502.4.16 branchAt() [2/6]

```
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]
```

15.502.4.17 branchAt() [3/6]

```
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]
```

15.502.4.18 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.502.4.19 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            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]
```

15.502.4.20 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

15.502.4.21 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in smtrat::PModule.

15.502.4.22 checkCore()

```
template<class 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.

Report unsatisfiability if the already found conflicts are still unresolved

Predicate that decides if the given conflict is a subset of the asserted constraints

Apply the method only if the asserted formula is not trivially undecidable

Update the linearization of all changed separators

Determine the direction that shall be active

Update the linearization if the direction has changed

Restrict the normal vector component of integral variables to positive values

Check the constructed linearization with the LRA solver

Learn the conflicting set of separators to avoid its recheck in the future

Check the asserted formula with the backends

Reimplemented from smtrat::Module.

15.502.4.23 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
    std::vector< FormulaSetT >::const_iterator _infsubset,
    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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.502.4.24 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.

15.502.4.25 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.502.4.26 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.502.4.27 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.502.4.28 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.502.4.29 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.502.4.30 collectOrigins() [1/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

15.502.4.31 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

_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.

15.502.4.32 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.502.4.33 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraintsToInform ( ) const [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.502.4.34 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

15.502.4.35 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)

15.502.4.36 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

_constraint	The constraint to remove from internal data structures.

15.502.4.37 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.502.4.38 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.502.4.39 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.502.4.40 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceived↩
VariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.502.4.41 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBest↩
Origin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.502.4.42 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.

15.502.4.43 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.

15.502.4.44 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

15.502.4.45 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

15.502.4.46 getBackendsAllModels()

`void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

15.502.4.47 getBackendsModel()

`void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

15.502.4.48 getInfeasibleSubsets() [1/2]

`void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

15.502.4.49 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.502.4.50 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.502.4.51 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.

15.502.4.52 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.502.4.53 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.502.4.54 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.502.4.55 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 reveived formula, if the Boolean is true.

Reimplemented in smtrat::PModule.

15.502.4.56 hasLemmas()

```
bool smtrat::Module::hasLemmas ( ) [inline], [inherited]
```

Checks whether this module or any of its backends provides any lemmas.

15.502.4.57 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).

15.502.4.58 id()

```
std::size_t smtrat::Module::id ( ) const [inline], [inherited]
```

Returns A unique ID to identify this module instance.

15.502.4.59 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.

15.502.4.60 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.502.4.61 informBackends()

```
void smtrat::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.

15.502.4.62 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, and smtrat::ICPModule< ICPSettings1 >.

15.502.4.63 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.502.4.64 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::FPPModule< Settings >.

15.502.4.65 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.502.4.66 isLemmaLevel() `bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.502.4.67 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().

15.502.4.68 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.

15.502.4.69 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.502.4.70 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.502.4.71 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

_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.

15.502.4.72 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.

15.502.4.73 objective() `carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

15.502.4.74 originInReceivedFormula() `bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

15.502.4.75 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula. TODO: disable this method

15.502.4.76 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula.

15.502.4.77 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to the backends of this module.

15.502.4.78 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to this module.

15.502.4.79 print()

```
void smtrat::Module::print (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.502.4.80 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

_out	The stream to print the assignment on.

15.502.4.81 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const  [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.502.4.82 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

_out	The stream to print the assignment on.

15.502.4.83 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const  [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.502.4.84 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const  [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.502.4.85 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.

15.502.4.86 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

15.502.4.87 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩
InfeasibleSubset (
            ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

15.502.4.88 receivedVariable()

```
bool smtrat::Module::receivedVariable (
            carl::Variable::Arg _var ) const [inline], [inherited]
```

15.502.4.89 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

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

15.502.4.90 removeCore()

```
template<class 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.

Normalize the left hand side of the constraint and turn the relation accordingly

Retrieve the normalized constraint and mark the separator object as changed

Check if the removed constraint prohibited the application of this method

Reimplemented from smtrat::Module.

15.502.4.91 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.502.4.92 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]
```

15.502.4.93 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.502.4.94 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.502.4.95 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]
```

Reimplemented in smtrat::PModule.

15.502.4.96 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

_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 in smtrat::PModule.

15.502.4.97 setId()

```
void smtrat::Module::setId ( std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.502.4.98 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.

15.502.4.99 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.

15.502.4.100 splitUnequalConstraint()

```
void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]
```

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.502.4.101 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.502.4.102 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 >.

15.502.4.103 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.502.4.104 updateModel() `template<class 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. Stores all informations retrieved from the LRA solver to construct the model

Retrieve the sign and exponent for every active variable

Calculate smallest possible integer valued exponents

Find model by increasingly testing the sample base

Reimplemented from smtrat::Module.

15.502.4.105 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).

15.502.5 Field Documentation

15.502.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.502.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.502.5.3 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.

15.502.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.502.5.5 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

15.502.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.502.5.7 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.

15.502.5.8 mFirstUncheckedReceivedSubformula `ModuleInput::const_iterator smtrat::Module::m↩FirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

15.502.5.9 mFoundAnswer `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.502.5.10 mFullCheck `bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.502.5.11 mInfeasibleSubsets `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.502.5.12 mInformedConstraints `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.502.5.13 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.

15.502.5.14 mLemmas `std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.502.5.15 mModel `Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.502.5.16 mModelComputed `bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.502.5.17 mNumOfBranchVarsToStore `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.502.5.18 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.

15.502.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]`

Reusable splitting variables.

15.502.5.20 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.502.5.21 mSmallerMusesCheckCounter

`unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.502.5.22 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.

15.502.5.23 mTheoryPropagations

`std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]`

15.502.5.24 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).

15.502.5.25 mVariableCounters

`std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.503 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.

15.503.1 Detailed Description

All properties needed to create a submit file.

15.503.2 Field Documentation

15.503.2.1 file_suffix `std::string benchmax::slurm::SubmitfileProperties::file_suffix`

Suffix for job and submit file.

15.503.2.2 filename_jobs `std::string benchmax::slurm::SubmitfileProperties::filename_jobs`

Filename of the job list file.

15.503.2.3 grace_time `carl::settings::duration benchmax::slurm::SubmitfileProperties::grace_time`

Grace time in seconds.

15.503.2.4 limit_memory `carl::settings::binary_quantity benchmax::slurm::SubmitfileProperties::limit_memory`

Memory limit in megabytes.

15.503.2.5 limit_time `carl::settings::duration benchmax::slurm::SubmitfileProperties::limit_time`

Time limit in seconds.

15.503.2.6 slices `std::size_t benchmax::slurm::SubmitfileProperties::slices`

Number of slices.

15.503.2.7 tasks `std::size_t benchmax::slurm::SubmitfileProperties::tasks`

Number of tasks.

15.503.2.8 tmp_dir `std::string benchmax::slurm::SubmitfileProperties::tmp_dir`

Temporary directory for log files.

15.504 smtrat::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 > &&_o↩ Conditions, smtrat::ConstraintsT &&_sideCondition=smtrat::ConstraintsT())

  Constructs a substitution with no square root term to map to.
- Substitution (const carl::Variable &, const smtrat::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.
- Substitution (const Substitution &_substitution)

  Copy constructor.
- ~Substitution ()

  The destructor.
- const carl::Variable & variable () const
- const smtrat::SqrtEx & term () const
- void setTerm (const smtrat::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 smtrat::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.

15.504.1 Member Enumeration Documentation

15.504.1.1 Type `enum smtrat::vs::Substitution::Type`

The type of a substitution.

Enumerator

NORMAL	
PLUS_EPSILON	
MINUS_INFINITY	
PLUS_INFINITY	
INVALID	

15.504.2 Constructor & Destructor Documentation

15.504.2.1 Substitution() [1/3]

```
smtrat::vs::Substitution::Substitution (
    const carl::Variable & _variable,
    const Type & _type,
    carl::PointerSet< Condition > && _oConditions,
    smtrat::ConstraintsT && _sideCondition = smtrat::ConstraintsT() )
```

Constructs a substitution with no square root term to map to.

Parameters

_variable	The variable to substitute of the substitution to construct.
_type	The type of the substitution of the substitution to construct.
_oConditions	The original conditions of the substitution to construct.
_sideCondition	The side conditions of the substitution to construct.

15.504.2.2 Substitution() [2/3]

```
smtrat::vs::Substitution::Substitution (
    const carl::Variable & _variable,
    const smtrat::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

_variable	The variable to substitute of the substitution to construct.
_term	The square root term to which the variable maps to.
_type	The type of the substitution of the substitution to construct.
_oConditions	The original conditions of the substitution to construct.
_sideCondition	The side conditions of the substitution to construct.

15.504.2.3 Substitution() [3/3]

```
smtrat::vs::Substitution::Substitution (
    const Substitution & _substitution )
```

Copy constructor.

15.504.2.4 ~Substitution() `smtrat::vs::Substitution::~Substitution ( )`

The destructor.

15.504.3 Member Function Documentation

15.504.3.1 operator==()

```
bool smtrat::vs::Substitution::operator== (
    const Substitution & _substitution ) const
```

Parameters

The	substitution to compare with.

Returns true, if this substitution is equal to the given substitution; false, otherwise.

15.504.3.2 originalConditions()

```
const carl::PointerSet<Condition>& smtrat::vs::Substitution↩
::originalConditions ( ) const [inline]
```

Returns A constant reference to the original conditions of this substitution.

15.504.3.3 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

The	substitution to compare with.

Returns true, if this substitution is less than the given substitution; false, otherwise.

Parameters

_withOrigins	A flag indicating whether to print also the origins of this substitution.
_withSideConditions	A flag indication whether to also the side conditions of this substitution.
_out	The stream to print on.
_init	The string to print at the beginning of every row.

15.504.3.4 rOriginalConditions()

```
carl::PointerSet<Condition>& smtrat::vs::Substitution::r↩
OriginalConditions ( ) [inline]
```

Returns A reference to the original conditions of this substitution.

15.504.3.5 rType() `Type& smtrat::vs::Substitution::rType ( ) [inline]`

Returns A reference to the type of this substitution.

15.504.3.6 setTerm() `void smtrat::vs::Substitution::setTerm ( const smtrat::Rational & _value ) [inline]`

Sets the substitution term to the given rational.

Parameters

_value	The value to set the substitution term to.

15.504.3.7 sideCondition() `const smtrat::ConstraintsT& smtrat::vs::Substitution::sideCondition ( ) const [inline]`

Returns A constant reference to the side condition of this substitution.

15.504.3.8 term() `const smtrat::SqrtEx& smtrat::vs::Substitution::term ( ) const [inline]`

Returns A constant reference to the term this substitution maps its variable to.

15.504.3.9 termVariables() `const carl::Variables& smtrat::vs::Substitution::termVariables ( ) const [inline]`

Returns A constant reference to the variables occurring in the substitution term.

15.504.3.10 type() `const Type& smtrat::vs::Substitution::type ( ) const [inline]`

Returns A constant reference to the type of this substitution.

15.504.3.11 valuate() `unsigned smtrat::vs::Substitution::valuate ( bool _preferMinusInf = true ) const`

Parameters

_preferMinusInf	A flag indicating whether to valuate the substitution type best or otherwise worst.

Returns The valuation of this substitution according to a heuristic.

15.504.3.12 variable() `const carl::Variable& smtrat::vs::Substitution::variable ( ) const [inline]`

Returns A constant reference to the variable this substitution substitutes.

15.505 smtrat::vs::substitutionPointerEqual Struct Reference

`#include <Substitution.h>`

Public Member Functions

- bool operator() (const Substitution *_substitutionA, const Substitution *_substitutionB) const

15.505.1 Member Function Documentation

15.505.1.1 operator()() `bool smtrat::vs::substitutionPointerEqual::operator() ( const Substitution * _substitutionA, const Substitution * _substitutionB ) const [inline]`

15.506 smtrat::vs::substitutionPointerHash Struct Reference

`#include <Substitution.h>`

Public Member Functions

- size_t operator() (const Substitution *_substitution) const

15.506.1 Member Function Documentation

15.506.1.1 operator()() `size_t smtrat::vs::substitutionPointerHash::operator() ( const Substitution * _substitution ) const [inline]`

15.507 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

15.507.1 Detailed Description This class is a `boost::spirit::qi` grammar that matches all kinds of smtlib symbols, numbers etc.

15.507.2 Constructor & Destructor Documentation

15.507.2.1 SymbolParser() `delta::SymbolParser::SymbolParser ( ) [inline]`

15.507.3 Field Documentation

15.507.3.1 main `qi::rule<Iterator, std::string(), Skipper> delta::SymbolParser::main`

15.507.3.2 quoted `qi::rule<Iterator, std::string(), Skipper> delta::SymbolParser::quoted`

15.507.3.3 simple `qi::rule<Iterator, std::string(), Skipper> delta::SymbolParser::simple`

15.508 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

15.508.1 Constructor & Destructor Documentation

15.508.1.1 SymbolParser() `smtrat::parser::SymbolParser::SymbolParser ( ) [inline]`

15.508.2 Field Documentation

15.508.2.1 main `qi::rule<Iterator, std::string(), Skipper> smtrat::parser::SymbolParser::main`

15.508.2.2 quoted `qi::rule<Iterator, std::string(), Skipper> smtrat::parser::SymbolParser↩ ::quoted`

15.508.2.3 simple `SimpleSymbolParser smtrat::parser::SymbolParser::simple`

15.509 smtrat::SymmetryModule< Settings > Class Template Reference

`#include <SymmetryModule.h>`

Public Types

- typedef Settings SettingsType
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::PERMANENT = 1 }

Public Member Functions

- SymmetryModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=nullptr)
- ~SymmetryModule ()
- 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 smtrat::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 _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.

- 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 &_branching↩ Value) 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 _preferLeft↩ Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩ ::vector< FormulaT >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩ AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩ CaseWeak=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.

15.509.1 Member Typedef Documentation

15.509.1.1 SettingsType `template<typename Settings >`

`typedef Settings smtrat::SymmetryModule< Settings >::SettingsType`

15.509.2 Member Enumeration Documentation

15.509.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

15.509.3 Constructor & Destructor Documentation

15.509.3.1 SymmetryModule() `template<class Settings >`

```
smtrat::SymmetryModule< Settings >::SymmetryModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

15.509.3.2 ~SymmetryModule() `template<class Settings >`

```
smtrat::SymmetryModule< Settings >::~SymmetryModule ( )
```

15.509.4 Member Function Documentation

15.509.4.1 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

_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.

15.509.4.2 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 >.

15.509.4.3 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

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 smtrat::SATModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::VSModule< Settings >, smtrat::PFEModule< Settings >, smtrat::EQModule< Setting smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Sett smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, and smtrat::GBModule< Settings >.

15.509.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.509.4.5 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.

15.509.4.6 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.

15.509.4.7 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.

15.509.4.8 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.

15.509.4.9 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.

15.509.4.10 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

_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.

15.509.4.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns All satisfying assignments, if existent.

15.509.4.12 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.

15.509.4.13 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.

15.509.4.14 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.

15.509.4.15 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.

15.509.4.16 branchAt() [1/6]

```
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]
```

15.509.4.17 branchAt() [2/6]

```
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]
```

15.509.4.18 branchAt() [3/6]

```
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]
```

15.509.4.19 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.509.4.20 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            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]
```

15.509.4.21 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

15.509.4.22 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

15.509.4.23 checkCore()

```
template<class Settings >
Answer smtrat::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 smtrat::Module.

15.509.4.24 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
    std::vector< FormulaSetT >::const_iterator _infsubset,
    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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.509.4.25 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.

15.509.4.26 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.509.4.27 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.509.4.28 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.509.4.29 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.509.4.30 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.509.4.31 collectOrigins() [1/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

15.509.4.32 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.

15.509.4.33 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.509.4.34 constraintsToInform() `const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.509.4.35 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

15.509.4.36 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)

15.509.4.37 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

_constraint	The constraint to remove from internal data structures.

15.509.4.38 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.509.4.39 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.509.4.40 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.509.4.41 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceived↩
VariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.509.4.42 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBest↩
Origin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.509.4.43 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.

15.509.4.44 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.

15.509.4.45 freeSplittingVariable() `static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

15.509.4.46 generateTrivialInfeasibleSubset() `void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

15.509.4.47 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

15.509.4.48 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

15.509.4.49 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

15.509.4.50 getInfeasibleSubsets() [2/2] `std::vector< FormulaSetT > smtrat::Module::getInfeasible↩ Subsets ( 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.509.4.51 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.509.4.52 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.

15.509.4.53 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.509.4.54 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.509.4.55 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.509.4.56 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 reveived formula, if the Boolean is true.

Reimplemented from smtrat::Module.

15.509.4.57 hasLemmas() bool smtrat::Module::hasLemmas ( ) [inline], [inherited]

Checks whether this module or any of its backends provides any lemmas.

15.509.4.58 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).

15.509.4.59 id() std::size_t smtrat::Module::id ( ) const [inline], [inherited]

Returns A unique ID to identify this module instance.

15.509.4.60 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.

15.509.4.61 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.509.4.62 informBackends()

```
void smtrat::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.

15.509.4.63 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, and smtrat::ICPModule< ICPSettings1 >.

15.509.4.64 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.509.4.65 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::FPPModule< Settings >.

15.509.4.66 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.509.4.67 isLemmaLevel() `bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.509.4.68 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().

15.509.4.69 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.

15.509.4.70 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.509.4.71 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.509.4.72 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.

15.509.4.73 moduleName() `virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]`

Returns The name of the given module type as name.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::EQPreprocessingModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::VSModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::GBModule< Settings >, smtrat::IntEqModule< Setti smtrat::CubeLIAModule< Settings >, smtrat::CurryModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IncWidthModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::BEModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::SplitSOSModule< Settings >.

15.509.4.74 objective() `carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

15.509.4.75 originInReceivedFormula() `bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

15.509.4.76 passedFormulaBegin() `ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula. TODO: disable this method

15.509.4.77 passedFormulaEnd() `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula.

15.509.4.78 pPassedFormula() `const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to the backends of this module.

15.509.4.79 pReceivedFormula() `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to this module.

15.509.4.80 print() `void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.509.4.81 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

_out	The stream to print the assignment on.

15.509.4.82 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.509.4.83 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

_out	The stream to print the assignment on.

15.509.4.84 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.509.4.85 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.509.4.86 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.

15.509.4.87 receivedFormulaChecked()

```
void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]
```

Notifies that the received formulas has been checked.

15.509.4.88 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩InfeasibleSubset (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

15.509.4.89 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const [inline], [inherited]
```

15.509.4.90 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

_subformula	The sub formula of the received formula to remove.

Reimplemented from smtrat::Module.

15.509.4.91 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

formula	The sub formula of the received formula to remove.

Reimplemented in smtrat::SATModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::VSModule< Settings >, smtrat::PFEModule< Settings >, smtrat::EQModule< Setting smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::CSplitModule< Settings >, smtrat::ICEModule< Settings >, smtrat::STropModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::PBPPModule< Sett smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::IntEqModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::IncWidthModule< Settings >, and smtrat::GBModule< Settings >.

15.509.4.92 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin
(
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.509.4.93 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

15.509.4.94 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline],
[inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.509.4.95 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.509.4.96 runBackends() [1/2] virtual Answer smtrat::PModule::runBackends ( ) [inline], [virtual], [inherited]

Reimplemented from smtrat::Module.

15.509.4.97 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.

15.509.4.98 setId() void smtrat::Module::setId (std::size_t _id) [inline], [inherited]

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.509.4.99 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.

15.509.4.100 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.

15.509.4.101 splitUnequalConstraint()

`void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.509.4.102 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.509.4.103 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 >.

15.509.4.104 updateLemmas()

`void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.509.4.105 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.

15.509.4.106 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).

15.509.5 Field Documentation

15.509.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.509.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.509.5.3 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.

15.509.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.509.5.5 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

15.509.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.509.5.7 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.

15.509.5.8 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.

15.509.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.509.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.509.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.509.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.509.5.13 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.

15.509.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.509.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.509.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.509.5.17 mNumOfBranchVarsToStore `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.509.5.18 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.

15.509.5.19 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

15.509.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.509.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.509.5.22 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.

15.509.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

15.509.5.24 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).

15.509.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.510 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 ()
- EvalRationalMap 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 nonbassi 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 efficienlty.

- 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 int &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 initital 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 &_horiIter, bool _horiIterLeftFromVertIter, Iterator &_vertIter, bool _vertIterBelowHoriIter)

  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

15.510.1 Constructor & Destructor Documentation

15.510.1.1 Tableau()

```
template<class Settings, typename T1, typename T2>
smtrat::lra::Tableau< Settings, T1, T2 >::Tableau (
            ModuleInput::iterator _defaultBoundPosition )
```

Parameters

_defaultBoundPosition	

15.510.1.2 ~Tableau()

```
template<class Settings, typename T1, typename T2>
smtrat::lra::Tableau< Settings, T1, T2 >::~Tableau ( )
```

15.510.2 Member Function Documentation

15.510.2.1 activateBasicVar()

```
template<class Settings, typename T1, typename T2>
void smtrat::lra::Tableau< Settings, T1, T2 >::activateBasicVar (
            Variable< T1, T2 > * _var )
```

g

Parameters

_var	

15.510.2.2 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	

15.510.2.3 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
```

15.510.2.4 addToEntry()

```
template<class Settings, typename T1, typename T2>
void smtrat::lra::Tableau< Settings, T1, T2 >::addToEntry (
            const T2 & _toAdd,
            Iterator & _horiIter,
            bool _horiIterLeftFromVertIter,
            Iterator & _vertIter,
            bool _vertIterBelowHoriIter )
```

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.
_horiIter	The iterator moving horizontally and, hence, giving the vertical position of the entry to add the given value to.
_horiIterLeftFromVertIter	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.
_vertIterBelowHoriIter	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.

15.510.2.5 betterEntry()

```
template<class Settings, typename T1, typename T2>
bool smtrat::lra::Tableau< Settings, T1, T2 >::betterEntry (
            EntryID _isBetter,
            EntryID _than ) const
```

Parameters

_isBetter	
_than	

Returns

15.510.2.6 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

_var	
_stopCriterium	

Returns

15.510.2.7 checkCorrectness()

```
template<class Settings, typename T1, typename T2>
size_t smtrat::lra::Tableau< Settings, T1, T2 >::checkCorrectness ( ) const
```

Returns

15.510.2.8 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

_rowVar	
premises	

15.510.2.9 columns()

```
template<class Settings, typename T1, typename T2>
const std::vector<Variable<T1, T2>*>& smtrat::lra::Tableau< Settings, T1, T2 >::columns ( )
const [inline]
```

Returns

15.510.2.10 compressRows()

```
template<class Settings, typename T1, typename T2>
void smtrat::lra::Tableau< Settings, T1, T2 >::compressRows ( )
```

15.510.2.11 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 efficienlty.

For this the fact is used, that VioSum is piecewise linear (the proof can be found in the SoI paper).

Parameters

candidates	Update candidates, IMPORTANT: Must be sorted with an stabloe sorting algorithm
nVar	The updated nonbasic variable
positive	Whether the positve or negative update values are checked

15.510.2.12 computeLeavingCandidates()

```
template<class Settings, typename T1, typename T2>
void smtrat::lra::Tableau< Settings, T1, T2 >::computeLeavingCandidates (
            const int & 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

i	: Index if entering (nonbasic) variable.
leaving_candidates	: Used as storage object for update candidates.

15.510.2.13 constraintToBound()

```
template<class Settings, typename T1, typename T2>
const carl::FastMap<FormulaT, std::vector<const Bound<T1, T2>*>*>& smtrat::lra::Tableau<
Settings, T1, T2 >::constraintToBound ( ) const  [inline]
```

Returns

15.510.2.14 currentDelta()

```
template<class Settings, typename T1, typename T2>
const T1& smtrat::lra::Tableau< Settings, T1, T2 >::currentDelta ( ) const  [inline]
```

15.510.2.15 deactivateBasicVar()

```
template<class Settings, typename T1, typename T2>
void smtrat::lra::Tableau< Settings, T1, T2 >::deactivateBasicVar (
            Variable< T1, T2 > * _var )
```

Parameters

_var	

15.510.2.16 defaultBoundPosition()

```
template<class Settings, typename T1, typename T2>
auto smtrat::lra::Tableau< Settings, T1, T2 >::defaultBoundPosition ( ) const  [inline]
```

Returns

15.510.2.17 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	

15.510.2.18 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

15.510.2.19 entryIsNegative()

```
template<class Settings, typename T1, typename T2>
bool smtrat::lra::Tableau< Settings, T1, T2 >::entryIsNegative (
            const TableauEntry< T1, T2 > & _entry ) const  [inline]
```

15.510.2.20 entryIsPositive()

```
template<class Settings, typename T1, typename T2>
bool smtrat::lra::Tableau< Settings, T1, T2 >::entryIsPositive (
            const TableauEntry< T1, T2 > & _entry ) const  [inline]
```

15.510.2.21 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.

15.510.2.22 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

15.510.2.23 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

15.510.2.24 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.

15.510.2.25 getMultilineConflict()

```
template<class Settings, typename T1, typename T2>
std::vector< const Bound<T1, T2>* > smtrat::lra::Tableau< Settings, T1, T2 >::getMultiline↩
Conflict ( )
```

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.

15.510.2.26 getNumberOfEntries()

```
template<class Settings, typename T1, typename T2>
size_t smtrat::lra::Tableau< Settings, T1, T2 >::getNumberOfEntries (
            Variable< T1, T2 > * _rowVar )
```

Parameters

_rowVar	

Returns number of entries in the row belonging to _rowVar

15.510.2.27 getObjectiveVariable()

```
template<class Settings, typename T1, typename T2>
Variable<T1,T2>* smtrat::lra::Tableau< Settings, T1, T2 >::getObjectiveVariable (
            const Poly & _lhs )
```

15.510.2.28 getRationalAssignment()

```
template<class Settings, typename T1, typename T2>
EvalRationalMap smtrat::lra::Tableau< Settings, T1, T2 >::getRationalAssignment ( ) const
```

Returns

15.510.2.29 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 )
```

15.510.2.30 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

_ass	
_rowVar	

Returns NULL, if the cut can´t be constructed; otherwise the valid constraint is returned.

15.510.2.31 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.

15.510.2.32 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 occured.

15.510.2.33 isActive()

```
template<class Settings, typename T1, typename T2>
bool smtrat::lra::Tableau< Settings, T1, T2 >::isActive (
            const Variable< T1, T2 > & _var ) const  [inline]
```

15.510.2.34 isConflicting()

```
template<class Settings, typename T1, typename T2>
bool smtrat::lra::Tableau< Settings, T1, T2 >::isConflicting ( ) const
```

15.510.2.35 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

_basicVar	
_forIncreasingAssignment	

Returns

15.510.2.36 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

_basicVar	
_forIncreasingAssignment	

Returns

15.510.2.37 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

_poly	
_originalVars	
_isInteger	

Returns

15.510.2.38 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

_constraint	

Returns

15.510.2.39 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

_poly	
_isInteger	

Returns

15.510.2.40 newTableauEntry()

```
template<class Settings, typename T1, typename T2>
EntryID smtrat::lra::Tableau< Settings, T1, T2 >::newTableauEntry (
            const T2 & _content )
```

Parameters

_content	

Returns

15.510.2.41 nextPivotingElement()

```
template<class Settings, typename T1, typename T2>
std::pair<EntryID, bool> smtrat::lra::Tableau< Settings, T1, T2 >::nextPivotingElement ( )
```

Returns

15.510.2.42 nextPivotingElementForOptimizing()

```
template<class Settings, typename T1, typename T2>
std::pair<EntryID,bool> smtrat::lra::Tableau< Settings, T1, T2 >::nextPivotingElementFor↩
Optimizing (
            const Variable< T1, T2 > & _objective )
```

Returns

15.510.2.43 nextPivotingElementInfeasibilities()

```
template<class Settings, typename T1, typename T2>
std::pair<EntryID, bool> smtrat::lra::Tableau< Settings, T1, T2 >::nextPivotingElement↩
Infeasibilities ( )
```

Method for next pivoting element according to ”Simplex with Sum of Infeasibilities for SMT” by Tim King and Clark Barrett.

Herefore, one counts the vialation of every bound and searches for elements who minimize the violation sum of the next iteration.

Returns

15.510.2.44 nextZeroPivotingElementForOptimizing()

```
template<class Settings, typename T1, typename T2>
std::pair<EntryID,bool> smtrat::lra::Tableau< Settings, T1, T2 >::nextZeroPivotingElement↩
ForOptimizing (
            const Variable< T1, T2 > & _objective ) const
```

15.510.2.45 numberOfPivotingSteps()

```
template<class Settings, typename T1, typename T2>
size_t smtrat::lra::Tableau< Settings, T1, T2 >::numberOfPivotingSteps ( ) const  [inline]
```

Returns

15.510.2.46 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

15.510.2.47 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

15.510.2.48 pivot()

```
template<class Settings, typename T1, typename T2>
Variable<T1, T2>* smtrat::lra::Tableau< Settings, T1, T2 >::pivot (
            EntryID _pivotingElement,
            bool _optimizing = false )
```

Parameters

_pivotingElement	
updateAssignments	

Returns

15.510.2.49 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

_pivotingElement	
_out	
_init	
_friendlyNames	
_withZeroColumns	

15.510.2.50 printEntries()

```
template<class Settings, typename T1, typename T2>
void smtrat::lra::Tableau< Settings, T1, T2 >::printEntries ( ) [inline]
```

15.510.2.51 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	

15.510.2.52 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	

15.510.2.53 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	

15.510.2.54 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	

15.510.2.55 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

_allBounds	
_out	
_init	

15.510.2.56 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

15.510.2.57 removeBound()

```
template<class Settings, typename T1, typename T2>
void smtrat::lra::Tableau< Settings, T1, T2 >::removeBound (
            const FormulaT & _constraint )
```

15.510.2.58 removeEntry()

```
template<class Settings, typename T1, typename T2>
void smtrat::lra::Tableau< Settings, T1, T2 >::removeEntry (
            EntryID _entryID )
```

Parameters

_entryID	

15.510.2.59 resetAssignment()

```
template<class Settings, typename T1, typename T2>
void smtrat::lra::Tableau< Settings, T1, T2 >::resetAssignment ( )
```

15.510.2.60 resetNumberOfPivotingSteps()

```
template<class Settings, typename T1, typename T2>
void smtrat::lra::Tableau< Settings, T1, T2 >::resetNumberOfPivotingSteps ( ) [inline]
```

15.510.2.61 resetTheta()

```
template<class Settings, typename T1, typename T2>
void smtrat::lra::Tableau< Settings, T1, T2 >::resetTheta ( ) [inline]
```

15.510.2.62 rLearnedLowerBounds()

```
template<class Settings, typename T1, typename T2>
carl::FastPointerMap<Variable<T1,T2>, LearnedBound>& smtrat::lra::Tableau< Settings, T1, T2
>::rLearnedLowerBounds ( ) [inline]
```

Returns

15.510.2.63 rLearnedUpperBounds()

```
template<class Settings, typename T1, typename T2>
carl::FastPointerMap<Variable<T1,T2>, LearnedBound>& smtrat::lra::Tableau< Settings, T1, T2
>::rLearnedUpperBounds ( ) [inline]
```

Returns

15.510.2.64 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

15.510.2.65 rowCorrect()

```
template<class Settings, typename T1, typename T2>
bool smtrat::lra::Tableau< Settings, T1, T2 >::rowCorrect (
            size_t _rowNumber ) const
```

Parameters

_rowNumber	

Returns

15.510.2.66 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

_basicVar	The basic variable for which to refine the supremum and infimum.

15.510.2.67 rows()

```
template<class Settings, typename T1, typename T2>
const std::vector<Variable<T1, T2>*>& smtrat::lra::Tableau< Settings, T1, T2 >::rows ( )
const [inline]
```

Returns

15.510.2.68 setBlandsRuleStart()

```
template<class Settings, typename T1, typename T2>
void smtrat::lra::Tableau< Settings, T1, T2 >::setBlandsRuleStart (
            size_t _start ) [inline]
```

15.510.2.69 setInfeasibilityRow()

```
template<class Settings, typename T1, typename T2>
void smtrat::lra::Tableau< Settings, T1, T2 >::setInfeasibilityRow ( )
```

Method to set the initital infeasibility row to avoid recomputation in every round.

15.510.2.70 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

_expectedHeight	
_expectedWidth	
_expectedNumberOfBounds	

15.510.2.71 size()

```
template<class Settings, typename T1, typename T2>
size_t smtrat::lra::Tableau< Settings, T1, T2 >::size ( ) const [inline]
```

Returns

15.510.2.72 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

15.510.2.73 storeAssignment()

```
template<class Settings, typename T1, typename T2>
void smtrat::lra::Tableau< Settings, T1, T2 >::storeAssignment ( )
```

15.510.2.74 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

_var	
_stopCriterium	

Returns

15.510.2.75 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

_downwards	The flag indicating whether to update the tableau downwards or upwards starting from the pivoting row.
_pivotingElement	The id of the current pivoting element.
_pivotingRowLeftSide	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.
_pivotingRowRightSide	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.
_updateAssignments	If true, the assignments of all variables will be updated after pivoting.

15.510.2.76 updateBasicAssignments()

```
template<class Settings, typename T1, typename T2>
void smtrat::lra::Tableau< Settings, T1, T2 >::updateBasicAssignments (
            size_t _column,
            const Value< T1 > & _change )
```

Parameters

_column	
_change	

15.510.2.77 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

_pivotingElement	Element in the row of the nonbasic variable

15.510.2.78 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

_pivotingElement	Element in the row of the nonbasic variable
update	Update Value of non-basic var

15.510.2.79 usedBlandsRule()

```
template<class Settings, typename T1, typename T2>
bool smtrat::lra::Tableau< Settings, T1, T2 >::usedBlandsRule ( )
```

Returns true, if the next pivoting element is selected by blands rule.

15.510.2.80 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.

15.511 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 ()

15.511.1 Constructor & Destructor Documentation

15.511.1.1 TableauEntry() [1/3]

```
template<typename T1, typename T2>
smtrat::lra::TableauEntry< T1, T2 >::TableauEntry ( ) [inline]
```

15.511.1.2 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

_up	
_down	
_left	
_right	
_rowVar	
_columnVar	
_content	

15.511.1.3 TableauEntry() [3/3]

```
template<typename T1, typename T2>
smtrat::lra::TableauEntry< T1, T2 >::TableauEntry (
            const TableauEntry< T1, T2 > & _entry ) [inline]
```

Parameters

_entry	

15.511.1.4 ~TableauEntry()

```
template<typename T1, typename T2>
smtrat::lra::TableauEntry< T1, T2 >::~TableauEntry ( ) [inline]
```

15.511.2 Member Function Documentation

15.511.2.1 columnVar()

```
template<typename T1, typename T2>
Variable<T1, T2>* smtrat::lra::TableauEntry< T1, T2 >::columnVar ( ) const [inline]
```

Returns

15.511.2.2 content()

```
template<typename T1, typename T2>
const T2& smtrat::lra::TableauEntry< T1, T2 >::content ( ) const [inline]
```

Returns

15.511.2.3 hNext()

```
template<typename T1, typename T2>
EntryID smtrat::lra::TableauEntry< T1, T2 >::hNext (
            bool leftwards ) [inline]
```

Parameters

leftwards	

Returns

15.511.2.4 rContent()

```
template<typename T1, typename T2>
T2& smtrat::lra::TableauEntry< T1, T2 >::rContent ( ) [inline]
```

Returns

15.511.2.5 rowVar()

```
template<typename T1, typename T2>
Variable<T1, T2>* smtrat::lra::TableauEntry< T1, T2 >::rowVar ( ) const [inline]
```

Returns

15.511.2.6 setColumnVar()

```
template<typename T1, typename T2>
void smtrat::lra::TableauEntry< T1, T2 >::setColumnVar (
            Variable< T1, T2 > * _columnVar ) [inline]
```

Parameters

_columnVar	

15.511.2.7 setHNext()

```
template<typename T1, typename T2>
void smtrat::lra::TableauEntry< T1, T2 >::setHNext (
            bool leftwards,
            const EntryID _entryId ) [inline]
```

Parameters

leftwards	
_entryId	

15.511.2.8 setRowVar()

```
template<typename T1, typename T2>
void smtrat::lra::TableauEntry< T1, T2 >::setRowVar (
            Variable< T1, T2 > * _rowVar ) [inline]
```

Parameters

_rowVar	

15.511.2.9 setVNext()

```
template<typename T1, typename T2>
void smtrat::lra::TableauEntry< T1, T2 >::setVNext (
            bool downwards,
            const EntryID _entryId ) [inline]
```

Parameters

downwards	
_entryId	

15.511.2.10 vNext()

```
template<typename T1, typename T2>
EntryID smtrat::lra::TableauEntry< T1, T2 >::vNext (
            bool downwards ) [inline]
```

Parameters

downwards	

Returns

15.512 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

15.512.1 Detailed Description

Tagged Polynomials.

15.512.2 Field Documentation

15.512.2.1 deg

```
std::size_t smtrat::mcsat::onecellcad::TagPoly::deg = 0
```

15.512.2.2 level

```
std::size_t smtrat::mcsat::onecellcad::TagPoly::level
```

15.512.2.3 poly

```
Poly smtrat::mcsat::onecellcad::TagPoly::poly
```

15.512.2.4 tag

```
InvarianceType smtrat::mcsat::onecellcad::TagPoly::tag
```

15.513 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

15.513.1 Constructor & Destructor Documentation

15.513.1.1 Task()

```
template<typename T >
smtrat::Task::Task (
            T && task,
            const Module * module,
            std::size_t _index )  [inline]
```

15.513.2 Member Function Documentation

15.513.2.1 conditionalIndex()

```
std::size_t smtrat::Task::conditionalIndex ( ) const  [inline]
```

15.513.2.2 getFuture() `std::future<Answer> smtrat::Task::getFuture ( ) [inline]`

15.513.2.3 getModule() `const Module* smtrat::Task::getModule ( ) const [inline]`

15.513.2.4 operator<() `bool smtrat::Task::operator< ( const Task & rhs ) const`

15.513.2.5 run() `void smtrat::Task::run ( ) [inline]`

15.514 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.

15.514.1 Detailed Description

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

15.514.2 Constructor & Destructor Documentation

15.514.2.1 TempFilenameGenerator() `delta::TempFilenameGenerator::TempFilenameGenerator ( const std::string & prefix ) [inline]`

Constructor.

Parameters

prefix	Prefix for all filenames.

15.514.2.2 ~TempFilenameGenerator() `delta::TempFilenameGenerator::~TempFilenameGenerator ( ) [inline]`

Destructor.

15.514.3 Member Function Documentation

15.514.3.1 get() `std::string delta::TempFilenameGenerator::get ( ) [inline]`

Retrieve a filename for a temporary file that is not in use.

Returns Temporary filename.

15.514.3.2 put() `void delta::TempFilenameGenerator::put ( const std::string & temp ) [inline]`

Returns a filename to the pool of available filenames.

Parameters

temp	Temporary filename.

15.515 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 specconstant
- 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

15.515.1 Member Typedef Documentation

15.515.1.1 Converter `typedef conversion::VariantVariantConverter<types::TermType> smtrat::parser::TermParser:`

15.515.2 Constructor & Destructor Documentation

15.515.2.1 TermParser() `smtrat::parser::TermParser::TermParser ( Theories * theories ) [inline]`

15.515.3 Field Documentation

15.515.3.1 attribute `AttributeParser smtrat::parser::TermParser::attribute`

15.515.3.2 binding `qi::rule<Iterator, Skipper> smtrat::parser::TermParser::binding`

15.515.3.3 converter `Converter smtrat::parser::TermParser::converter`

15.515.3.4 main `qi::rule<Iterator, types::TermType(), Skipper> smtrat::parser::TermParser↩ ::main`

15.515.3.5 qualifiedidentifier `QualifiedIdentifierParser smtrat::parser::TermParser::qualifiedidentifier`

15.515.3.6 sortedvariable `SortedVariableParser smtrat::parser::TermParser::sortedvariable`

15.515.3.7 specconstant `SpecConstantParser smtrat::parser::TermParser::specconstant`

15.515.3.8 symbol `SymbolParser smtrat::parser::TermParser::symbol`

15.515.3.9 termop `qi::rule<Iterator, types::TermType(), Skipper> smtrat::parser::TermParser↩ ::termop`

15.515.3.10 theories `Theories* smtrat::parser::TermParser::theories`

15.516 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

15.516.1 Member Function Documentation

15.516.1.1 operator==()

```
bool smtrat::mcsat::vs::helper::TestCandidate::operator== (
    const TestCandidate & other ) const  [inline]
```

15.516.2 Field Documentation

15.516.2.1 side_condition `ConstraintsT smtrat::mcsat::vs::helper::TestCandidate::side_condition`

15.516.2.2 term `carl::vs::Term<Poly> smtrat::mcsat::vs::helper::TestCandidate::term`

15.517 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.

15.517.1 Detailed Description The Theories class combines all implemented theories and provides a single interface to interact with all theories at once.

15.517.2 Member Typedef Documentation

15.517.2.1 Modules `typedef boost::mpl::vector< CoreTheory* , ArithmeticTheory* , BitvectorTheory* , UninterpretedTheory* , BooleanEncodingTheory* >::type smtrat::parser::Theories::Modules`

15.517.3 Constructor & Destructor Documentation

15.517.3.1 Theories()

```
smtrat::parser::Theories::Theories (
    ParserState * state ) [inline]
```

15.517.3.2 ~Theories()

```
smtrat::parser::Theories::~Theories ( ) [inline]
```

15.517.4 Member Function Documentation

15.517.4.1 addConstants()

```
static void smtrat::parser::Theories::addConstants (
    qi::symbols< char, types::ConstType > & constants ) [inline], [static]
```

Collects constants from all theory modules.

15.517.4.2 addGlobalFormulas()

```
void smtrat::parser::Theories::addGlobalFormulas (
    FormulasT & formulas ) [inline]
```

15.517.4.3 addSimpleSorts()

```
static void smtrat::parser::Theories::addSimpleSorts (
    qi::symbols< char, carl::Sort > & sorts ) [inline], [static]
```

Collects simple sorts from all theory modules.

15.517.4.4 annotateTerm()

```
const auto& smtrat::parser::Theories::annotateTerm (
    const types::TermType & term,
    const std::vector< Attribute > & attributes ) [inline]
```

15.517.4.5 declareFunction()

```
void smtrat::parser::Theories::declareFunction (
    const std::string & name,
    const std::vector< carl::Sort > & args,
    const carl::Sort & sort ) [inline]
```

15.517.4.6 declareFunctionArgument()

```
types::VariableType smtrat::parser::Theories::declare↩
FunctionArgument (
    const std::pair< std::string, carl::Sort > & arg ) [inline]
```

15.517.4.7 declareVariable()

```
void smtrat::parser::Theories::declareVariable (
    const std::string & name,
    const carl::Sort & sort ) [inline]
```

15.517.4.8 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

15.517.4.9 functionCall()

```
types::TermType smtrat::parser::Theories::functionCall (
    const Identifier & identifier,
    const std::vector< types::TermType > & arguments ) [inline]
```

15.517.4.10 handleDistinct()

```
types::TermType smtrat::parser::Theories::handleDistinct (
    const std::vector< types::TermType > & arguments ) [inline]
```

15.517.4.11 handleITE()

```
types::TermType smtrat::parser::Theories::handleITE (
    const std::vector< types::TermType > & arguments ) [inline]
```

15.517.4.12 handleLet()

```
void smtrat::parser::Theories::handleLet (
    const std::string & symbol,
    const types::TermType & term ) [inline]
```

15.517.4.13 instantiate()

```
bool smtrat::parser::Theories::instantiate (
    const UserFunctionInstantiator & function,
    const types::VariableType & var,
    const types::TermType & repl,
    types::TermType & subject ) [inline]
```

15.517.4.14 instantiateUserFunction()

```
bool smtrat::parser::Theories::instantiateUserFunction (
    const UserFunctionInstantiator & function,
    const std::vector< types::TermType > & arguments,
    types::TermType & result,
    TheoryError & errors ) [inline]
```

15.517.4.15 popExpressionScope()

```
void smtrat::parser::Theories::popExpressionScope (
    std::size_t n ) [inline]
```

15.517.4.16 popScriptScope()

```
void smtrat::parser::Theories::popScriptScope (
    std::size_t n ) [inline]
```

15.517.4.17 pushExpressionScope()

```
void smtrat::parser::Theories::pushExpressionScope (
    std::size_t n ) [inline]
```

15.517.4.18 pushScriptScope()

```
void smtrat::parser::Theories::pushScriptScope (
    std::size_t n ) [inline]
```

15.517.4.19 resolveSymbol()

```
types::TermType smtrat::parser::Theories::resolveSymbol (
    const Identifier & identifier ) const [inline]
```

15.517.4.20 resolveVariable()

```
types::VariableType smtrat::parser::Theories::resolveVariable (
    const Identifier & identifier ) const [inline]
```

15.518 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)

15.518.1 Member Function Documentation

15.518.1.1 next()

```
TheoryError& smtrat::parser::TheoryError::next ( ) [inline]
```

15.518.1.2 operator()()

```
TheoryError& smtrat::parser::TheoryError::operator() (
    const std::string & theory ) [inline]
```

15.518.1.3 operator<<()

```
template<typename T >
TheoryError& smtrat::parser::TheoryError::operator<< (
    const T & t ) [inline]
```

15.518.2 Friends And Related Function Documentation

15.518.2.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const TheoryError & te ) [friend]
```

15.519 smtrat::mcsat::TheoryLevel Struct Reference

```
#include <MCSATMixin.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.

15.519.1 Field Documentation

15.519.1.1 decidedVariables `std::vector<Minisat::Var> smtrat::mcsat::TheoryLevel::decidedVariables`

Boolean variables univariate in this theory variable.

15.519.1.2 decisionLiteral `Minisat::Lit smtrat::mcsat::TheoryLevel::decisionLiteral = Minisat::lit_Undef`

Literal that assigns this theory variable.

15.519.1.3 variable `carl::Variable smtrat::mcsat::TheoryLevel::variable = carl::Variable::NO_VARIABLE`

Theory variable for this level.

15.520 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.

15.520.1 Constructor & Destructor Documentation

15.520.1.1 TheoryPropagation() [1/4]

```
smtrat::Module::TheoryPropagation::TheoryPropagation (
    FormulasT && _premise,
    const FormulaT & _conclusion ) [inline]
```

Constructor.

15.520.1.2 TheoryPropagation() [2/4]

```
smtrat::Module::TheoryPropagation::TheoryPropagation ( ) [delete]
```

15.520.1.3 TheoryPropagation() [3/4]

```
smtrat::Module::TheoryPropagation::TheoryPropagation (
    const TheoryPropagation & ) [delete]
```

15.520.1.4 TheoryPropagation() [4/4] smtrat::Module::TheoryPropagation::TheoryPropagation (TheoryPropagation && _toMove) [inline]

15.520.1.5 ~TheoryPropagation() smtrat::Module::TheoryPropagation::~TheoryPropagation ( ) [inline]

15.520.2 Member Function Documentation

15.520.2.1 operator=() [1/2] TheoryPropagation& smtrat::Module::TheoryPropagation::operator= (const TheoryPropagation & _toMove) [delete]

15.520.2.2 operator=() [2/2] TheoryPropagation& smtrat::Module::TheoryPropagation::operator= (TheoryPropagation && _toMove) [inline]

15.520.3 Field Documentation

15.520.3.1 mConclusion FormulaT smtrat::Module::TheoryPropagation::mConclusion

The propagated constraint.

15.520.3.2 mPremise FormulasT smtrat::Module::TheoryPropagation::mPremise

The constraints under which the propagated constraint holds.

15.521 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

15.521.1 Detailed Description

template<mcsat::VariableOrdering vot>
class smtrat::TheoryVarSchedulerStatic< vot >

Schedules theory variables statically.

Should not be used directly.

15.521.2 Constructor & Destructor Documentation

15.521.2.1 TheoryVarSchedulerStatic()

```
template<mcsat::VariableOrdering vot>
template<typename BaseModule >
smtrat::TheoryVarSchedulerStatic< vot >::TheoryVarSchedulerStatic (
            BaseModule & baseModule ) [inline], [explicit]
```

15.521.3 Member Function Documentation

15.521.3.1 attachClause()

```
void smtrat::VarSchedulerBase::attachClause (
            Minisat::CRef ) [inline], [inherited]
```

15.521.3.2 decreaseActivity()

```
void smtrat::VarSchedulerBase::decreaseActivity (
            Minisat::Var ) [inline], [inherited]
```

15.521.3.3 detachClause()

```
void smtrat::VarSchedulerBase::detachClause (
            Minisat::CRef ) [inline], [inherited]
```

15.521.3.4 empty()

```
template<mcsat::VariableOrdering vot>
bool smtrat::TheoryVarSchedulerStatic< vot >::empty ( ) [inline]
```

15.521.3.5 increaseActivity()

```
void smtrat::VarSchedulerBase::increaseActivity (
            Minisat::Var ) [inline], [inherited]
```

15.521.3.6 insert()

```
template<mcsat::VariableOrdering vot>
void smtrat::TheoryVarSchedulerStatic< vot >::insert (
            Minisat::Var var ) [inline]
```

15.521.3.7 level()

```
template<mcsat::VariableOrdering vot>
size_t smtrat::TheoryVarSchedulerStatic< vot >::level ( ) [inline]
```

Level of the next theory variable.

15.521.3.8 pop()

```
template<mcsat::VariableOrdering vot>
Minisat::Lit smtrat::TheoryVarSchedulerStatic< vot >::pop ( ) [inline]
```

15.521.3.9 print()

```
template<mcsat::VariableOrdering vot>
void smtrat::TheoryVarSchedulerStatic< vot >::print ( ) const [inline]
```

15.521.3.10 rebuild()

```
void smtrat::VarSchedulerBase::rebuild ( ) [inline], [inherited]
```

Rebuild heap.

15.521.3.11 rebuildActivities()

```
void smtrat::VarSchedulerBase::rebuildActivities ( ) [inline],
[inherited]
```

15.521.3.12 rebuildTheoryVars()

```
template<mcsat::VariableOrdering vot>
template<typename Constraints >
void smtrat::TheoryVarSchedulerStatic< vot >::rebuildTheoryVars (
            const Constraints & c ) [inline]
```

15.521.3.13 relocateClauses()

```
void smtrat::VarSchedulerBase::relocateClauses (
            std::function< void(Minisat::CRef &)> ) [inline], [inherited]
```

15.521.3.14 univariateLevel()

```
template<mcsat::VariableOrdering vot>
size_t smtrat::TheoryVarSchedulerStatic< vot >::univariateLevel (
            Minisat::Var v ) [inline]
```

Returns the level in which the given constraint is univariate.

15.521.4 Field Documentation

15.521.4.1 abstractLiteral

`std::function<const Minisat::Lit(const FormulaT&)> smtrat::VarSchedulerBase::abstractLiteral [protected], [inherited]`

15.521.4.2 abstractVariable

`std::function< Minisat::Var(const FormulaT&)> smtrat::VarSchedulerBase::abstractVariable [protected], [inherited]`

15.521.4.3 carlVar

`std::function<carl::Variable(Minisat::Var)> smtrat::VarSchedulerMcsatBase::carlVar [protected], [inherited]`

15.521.4.4 currentlySatisfiedByBackend

`std::function<unsigned(const FormulaT&)> smtrat::VarSchedulerBase::currentlySatisfiedByBackend [protected], [inherited]`

15.521.4.5 currentModel

`std::function<Model()> smtrat::VarSchedulerMcsatBase::currentModel [protected], [inherited]`

15.521.4.6 getActivity

`std::function<double(Minisat::Var)> smtrat::VarSchedulerBase::getActivity [protected], [inherited]`

15.521.4.7 getBoolLitValue

`std::function<Minisat::lbool(Minisat::Lit)> smtrat::VarSchedulerBase::getBoolLitValue [protected], [inherited]`

15.521.4.8 getBoolVarValue

`std::function<Minisat::lbool(Minisat::Var)> smtrat::VarSchedulerBase::getBoolVarValue [protected], [inherited]`

15.521.4.9 getClause

`std::function<const Minisat::Clause&(Minisat::CRef)> smtrat::VarSchedulerBase::getClause [protected], [inherited]`

15.521.4.10 getPolarity

`std::function<char(Minisat::Var)> smtrat::VarSchedulerBase::getPolarity [protected], [inherited]`

15.521.4.11 isAbstractedFormula

`std::function<bool(const FormulaT&)> smtrat::VarSchedulerBase::isAbstractedFormula [protected], [inherited]`

15.521.4.12 isBoolValueUndef

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isBoolValueUndef [protected], [inherited]`

15.521.4.13 isDecisionVar `std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isDecisionVar [protected], [inherited]`

15.521.4.14 isTheoryAbstraction `std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isTheoryAbstraction [protected], [inherited]`

15.521.4.15 isTheoryVar `std::function<bool(Minisat::Var)> smtrat::VarSchedulerMcsatBase::isTheoryVar [protected], [inherited]`

15.521.4.16 minisatVar `std::function<Minisat::Var(carl::Variable)> smtrat::VarSchedulerMcsatBase::minisatVar [protected], [inherited]`

15.521.4.17 reabstractLiteral `std::function<const FormulaT&(Minisat::Lit)> smtrat::VarSchedulerBase::reabstractLiteral [protected], [inherited]`

15.521.4.18 reabstractVariable `std::function<const FormulaT&(Minisat::Var)> smtrat::VarSchedulerBase::reabstractVariable [protected], [inherited]`

15.521.4.19 setPolarity `std::function<void(Minisat::Var,bool)> smtrat::VarSchedulerBase::setPolarity [protected], [inherited]`

15.522 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)

15.522.1 Constructor & Destructor Documentation

15.522.1.1 ThreadPool()

```
smtrat::ThreadPool::ThreadPool (
    std::size_t maxThreads ) [inline]
```

15.522.1.2 ~ThreadPool()

`smtrat::ThreadPool::~ThreadPool ( ) [inline]`

15.522.2 Member Function Documentation

15.522.2.1 runBackends()

```
Answer smtrat::ThreadPool::runBackends (
    const std::vector< Module * > & _modules,
    bool _final,
    bool _full,
    carl::Variable _objective )
```

Parameters

_modules	
_final	
_full	
_objective	

15.523 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

15.523.1 Member Function Documentation

15.523.1.1 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.

15.523.1.2 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.

15.523.1.3 layout()

```
virtual void smtrat::cad::debug::TikzBasePrinter::layout ( ) [pure virtual]
```

Implemented in smtrat::cad::debug::TikzTreePrinter, and smtrat::cad::debug::TikzDAGPrinter.

15.523.1.4 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]
```

15.523.1.5 step()

```
void smtrat::cad::debug::TikzBasePrinter::step ( )  [inline]
```

15.523.1.6 width()

```
virtual std::size_t smtrat::cad::debug::TikzBasePrinter::width ( ) const
[pure virtual]
```

Implemented in smtrat::cad::debug::TikzTreePrinter, and smtrat::cad::debug::TikzDAGPrinter.

15.523.1.7 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.

15.523.2 Field Documentation

15.523.2.1 mStep

```
std::size_t smtrat::cad::debug::TikzBasePrinter::mStep  [protected]
```

15.524 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

15.524.1 Member Function Documentation

15.524.1.1 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.

15.524.1.2 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.

15.524.1.3 layout()

```
void smtrat::cad::debug::TikzDAGPrinter::layout ( ) [inline], [override], [virtual]
```

Implements smtrat::cad::debug::TikzBasePrinter.

15.524.1.4 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]
```

15.524.1.5 step()

```
void smtrat::cad::debug::TikzBasePrinter::step ( ) [inline], [inherited]
```

15.524.1.6 width()

```
std::size_t smtrat::cad::debug::TikzDAGPrinter::width ( ) const [inline], [override], [virtual]
```

Implements smtrat::cad::debug::TikzBasePrinter.

15.524.1.7 writeTo()

```
void smtrat::cad::debug::TikzDAGPrinter::writeTo (
    std::ostream & os,
    int xBase ) const [inline], [override], [virtual]
```

Implements smtrat::cad::debug::TikzBasePrinter.

15.524.2 Field Documentation

15.524.2.1 mStep

```
std::size_t smtrat::cad::debug::TikzBasePrinter::mStep [protected], [inherited]
```

15.525 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

15.525.1 Member Function Documentation

15.525.1.1 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]
```

15.525.1.2 addLifting()

```
template<typename L >
void smtrat::cad::debug::TikzHistoryPrinter::addLifting (
            const L & l )  [inline]
```

15.525.1.3 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]
```

15.525.1.4 addProjection()

```
template<typename P >
void smtrat::cad::debug::TikzHistoryPrinter::addProjection (
            const P & p )  [inline]
```

15.525.1.5 configure()

```
template<typename Printer >
void smtrat::cad::debug::TikzHistoryPrinter::configure (
            const std::string & name )  [inline]
```

15.525.1.6 layout()

```
void smtrat::cad::debug::TikzHistoryPrinter::layout ( )  [inline]
```

15.525.1.7 step()

```
void smtrat::cad::debug::TikzHistoryPrinter::step ( )  [inline]
```

15.525.1.8 writeTo()

```
void smtrat::cad::debug::TikzHistoryPrinter::writeTo (
            const std::string & filename ) const  [inline]
```

15.526 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

15.526.1 Member Function Documentation

15.526.1.1 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.

15.526.1.2 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.

15.526.1.3 layout()

```
void smtrat::cad::debug::TikzTreePrinter::layout ( ) [inline], [override],
[virtual]
```

Implements smtrat::cad::debug::TikzBasePrinter.

15.526.1.4 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]
```

15.526.1.5 setRoot()

```
void smtrat::cad::debug::TikzTreePrinter::setRoot (
    const std::string & node,
    const std::string & data ) [inline]
```

15.526.1.6 step() `void smtrat::cad::debug::TikzBasePrinter::step ( ) [inline], [inherited]`

15.526.1.7 width() `virtual std::size_t smtrat::cad::debug::TikzTreePrinter::width ( ) const [inline], [override], [virtual]`

Implements smtrat::cad::debug::TikzBasePrinter.

15.526.1.8 writeTo() `void smtrat::cad::debug::TikzTreePrinter::writeTo ( std::ostream & os, int xBase ) const [inline], [override], [virtual]`

Implements smtrat::cad::debug::TikzBasePrinter.

15.526.2 Field Documentation

15.526.2.1 mStep `std::size_t smtrat::cad::debug::TikzBasePrinter::mStep [protected], [inherited]`

15.527 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.

15.527.1 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.

15.527.2 Constructor & Destructor Documentation

15.527.2.1 Tool() [1/5]

```
benchmax::Tool::Tool ( ) [delete]
```

15.527.2.2 Tool() [2/5]

```
benchmax::Tool::Tool (
    const Tool & ) [delete]
```

15.527.2.3 Tool() [3/5]

```
benchmax::Tool::Tool (
    Tool && ) [delete]
```

15.527.2.4 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.

15.527.2.5 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.

15.527.2.6 ~Tool()

```
virtual benchmax::Tool::~Tool ( ) [virtual], [default]
```

15.527.3 Member Function Documentation

15.527.3.1 additionalResults()

```
virtual void benchmax::Tool::additionalResults (
    const fs::path & ,
    BenchmarkResult & ) const [inline], [virtual]
```

Recover additional results from the tool output.

Reimplemented in benchmax::SMTRAT, benchmax::Minisatp, benchmax::Minisat, benchmax::MathSAT, benchmax::SMTRAT_Analyzer, and benchmax::Z3.

15.527.3.2 attributeHash()

```
std::size_t benchmax::Tool::attributeHash ( ) const [inline]
```

Hash of the attributes.

15.527.3.3 attributes()

```
const std::map<std::string,std::string>& benchmax::Tool::attributes ( )
const [inline]
```

A set of attributes, for example compilation options.

15.527.3.4 binary()

```
fs::path benchmax::Tool::binary ( ) const [inline]
```

Full path to the binary.

15.527.3.5 canHandle()

```
virtual bool benchmax::Tool::canHandle (
    const fs::path & ) const [inline], [virtual]
```

Checks whether this cool can handle this file type.

Reimplemented in benchmax::SMTRAT, benchmax::Minisatp, benchmax::Minisat, benchmax::MathSAT, benchmax::SMTRAT_Analyzer, benchmax::SMTRAT_OPB, and benchmax::Z3.

15.527.3.6 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.

15.527.3.7 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.

15.527.3.8 name() `std::string benchmax::Tool::name ( ) const [inline]`

Common name of this tool.

15.527.3.9 operator=() [1/2] `Tool& benchmax::Tool::operator= ( const Tool & ) [delete]`

15.527.3.10 operator=() [2/2] `Tool& benchmax::Tool::operator= ( Tool && ) [delete]`

15.527.3.11 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.

15.527.3.12 resolveDependencies() `std::vector<std::string> benchmax::Tool::resolveDependencies ( ) const [inline]`

Get dependencies of binary required to run it (via ldd)

15.527.4 Friends And Related Function Documentation

15.527.4.1 operator< `bool operator< ( const Tool & lhs, const Tool & rhs ) [friend]`

Compare two tools.

15.527.5 Field Documentation

15.527.5.1 mArguments `std::string benchmax::Tool::mArguments [protected]`

Command line arguments that should be passed to the binary.

15.527.5.2 mAttributes `std::map<std::string,std::string> benchmax::Tool::mAttributes [protected]`

Attributes of the tool obtained by introspection of the binary.

15.527.5.3 mBinary `fs::path benchmax::Tool::mBinary [protected]`

Path to the binary.

15.527.5.4 mName `std::string benchmax::Tool::mName [protected]`

(Non-unique) identifier for the tool.

15.528 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_smtrat

  SMT-RAT with SMT-LIB interface.
- std::vector< std::filesystem::path > tools_smtrat_opb

  SMT-RAT with OPB interface.
- std::vector< std::filesystem::path > tools_smtrat_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.

15.528.1 Detailed Description

Tool-related settings.

15.528.2 Field Documentation

15.528.2.1 collect_statistics `bool benchmax::settings::ToolSettings::collect_statistics`

Whether or not to collect statistics.

15.528.2.2 tools_common_prefix `std::filesystem::path benchmax::settings::ToolSettings::tools_common_prefix`

Common prefix of tool binaries to simplify output.

15.528.2.3 tools_generic `std::vector<std::filesystem::path> benchmax::settings::ToolSettings::tools_generic`

Generic tools.

15.528.2.4 tools_mathsat `std::vector<std::filesystem::path> benchmax::settings::ToolSettings↩ ::tools_mathsat`

MathSAT with SMT-LIB interface.

15.528.2.5 tools_minisat `std::vector<std::filesystem::path> benchmax::settings::ToolSettings↩ ::tools_minisat`

Minisat with DIMACS interface.

15.528.2.6 tools_minisatp `std::vector<std::filesystem::path> benchmax::settings::ToolSettings↩ ::tools_minisatp`

Minisatp with OPB interface.

15.528.2.7 tools_smtrat `std::vector<std::filesystem::path> benchmax::settings::ToolSettings↩ ::tools_smtrat`

SMT-RAT with SMT-LIB interface.

15.528.2.8 tools_smtrat_analyzer `std::vector<std::filesystem::path> benchmax::settings::Tool↩ Settings::tools_smtrat_analyzer`

SMT-RAT formula analyzer.

15.528.2.9 tools_smtrat_opb `std::vector<std::filesystem::path> benchmax::settings::Tool↩ Settings::tools_smtrat_opb`

SMT-RAT with OPB interface.

15.528.2.10 tools_z3 `std::vector<std::filesystem::path> benchmax::settings::ToolSettings↩ ::tools_z3`

z3 with SMT-LIB interface.

15.529 smtrat::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.

```
#include <TotalizerEncoder.h>
```

Public Member Functions

- TotalizerEncoder ()
- ~TotalizerEncoder ()
- bool canEncode (const ConstraintT &constraint)
- Rational encodingSize (const ConstraintT &constraint)
- std::string name ()
- boost::optional< FormulaT > encode (const ConstraintT &constraint)

  Encodes an arbitrary constraint.

Data Fields

- std::size_t problem_size

Protected Member Functions

- boost::optional< FormulaT > doEncode (const ConstraintT &constraint)
- FormulaT generateVarChain (const std::set< carl::Variable > &vars, carl::FormulaType type)

15.529.1 Detailed Description 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.

15.529.2 Constructor & Destructor Documentation

15.529.2.1 TotalizerEncoder()

```
smtrat::TotalizerEncoder::TotalizerEncoder ( ) [inline]
```

15.529.2.2 ~TotalizerEncoder()

```
smtrat::TotalizerEncoder::~TotalizerEncoder ( )
```

15.529.3 Member Function Documentation

15.529.3.1 canEncode()

```
bool smtrat::TotalizerEncoder::canEncode (
    const ConstraintT & constraint ) [virtual]
```

Implements smtrat::PseudoBoolEncoder.

15.529.3.2 doEncode()

```
boost::optional< FormulaT > smtrat::TotalizerEncoder::doEncode (
    const ConstraintT & constraint ) [protected], [virtual]
```

Implements smtrat::PseudoBoolEncoder.

15.529.3.3 encode()

```
boost::optional< FormulaT > smtrat::PseudoBoolEncoder::encode (
    const ConstraintT & constraint ) [inherited]
```

Encodes an arbitrary constraint.

Returns encoded formula

15.529.3.4 encodingSize()

```
Rational smtrat::TotalizerEncoder::encodingSize (
    const ConstraintT & constraint ) [virtual]
```

Reimplemented from smtrat::PseudoBoolEncoder.

15.529.3.5 generateVarChain()

```
FormulaT smtrat::PseudoBoolEncoder::generateVarChain (
    const std::set< carl::Variable > & vars,
    carl::FormulaType type ) [protected], [inherited]
```

15.529.3.6 name()

```
std::string smtrat::TotalizerEncoder::name ( ) [inline], [virtual]
```

Reimplemented from smtrat::PseudoBoolEncoder.

15.529.4 Field Documentation

15.529.4.1 problem_size

```
std::size_t smtrat::PseudoBoolEncoder::problem_size [inherited]
```

15.530 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 ()

15.530.1 Constructor & Destructor Documentation

15.530.1.1 TotalizerTree()

```
smtrat::TotalizerTree::TotalizerTree (
    const std::set< carl::Variable > & variables )
```

15.530.1.2 ~TotalizerTree()

```
smtrat::TotalizerTree::~TotalizerTree ( ) [inline]
```

15.530.2 Member Function Documentation

15.530.2.1 isLeaf()

```
bool smtrat::TotalizerTree::isLeaf ( ) [inline]
```

15.530.2.2 left()

```
TotalizerTree& smtrat::TotalizerTree::left ( ) [inline]
```

15.530.2.3 right()

```
TotalizerTree& smtrat::TotalizerTree::right ( ) [inline]
```

15.530.2.4 variables()

```
std::vector<carl::Variable> smtrat::TotalizerTree::variables ( ) [inline]
```

15.531 smtrat::cad::projection_compare::type Struct Reference

```
#include <ProjectionComparator.h>
```

15.532 smtrat::cad::sample_compare::type Struct Reference

```
#include <SampleComparator.h>
```

15.533 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

15.533.1 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.

15.533.2 Member Typedef Documentation

15.533.2.1 value_type

```
template<typename T , bool compression>
typedef T smtrat::uf_impl::uf_data_wrapper< T, compression >::value_type
```

15.533.3 Constructor & Destructor Documentation

15.533.3.1 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]
```

15.533.4 Member Function Documentation

15.533.4.1 class_()

```
template<typename T , bool compression>
std::size_t smtrat::uf_impl::uf_data_wrapper< T, compression >::class_ ( ) const [inline],
[noexcept]
```

15.533.4.2 increment_rank()

```
template<typename T , bool compression>
void smtrat::uf_impl::uf_data_wrapper< T, compression >::increment_rank ( ) [inline], [noexcept]
```

15.533.4.3 rank()

```
template<typename T , bool compression>
std::size_t smtrat::uf_impl::uf_data_wrapper< T, compression >::rank ( ) const [inline], [noexcept]
```

15.533.4.4 set_class()

```
template<typename T , bool compression>
void smtrat::uf_impl::uf_data_wrapper< T, compression >::set_class (
            std::size_t newclass ) [inline], [noexcept]
```

15.533.4.5 set_rank()

```
template<typename T , bool compression>
void smtrat::uf_impl::uf_data_wrapper< T, compression >::set_rank (
            std::size_t rank_ ) [inline], [noexcept]
```

15.533.4.6 value() [1/2]

```
template<typename T , bool compression>
const T& smtrat::uf_impl::uf_data_wrapper< T, compression >::value ( ) const [inline], [noexcept]
```

15.533.4.7 value() [2/2]

```
template<typename T , bool compression>
T& smtrat::uf_impl::uf_data_wrapper< T, compression >::value ( ) [inline], [noexcept]
```

15.534 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

15.534.1 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.

15.534.2 Constructor & Destructor Documentation

15.534.2.1 uf_data_wrapper()

```
template<bool compression>
smtrat::uf_impl::uf_data_wrapper< void, compression >::uf_data_wrapper (
            std::size_t class_index ) [inline]
```

15.534.3 Member Function Documentation

15.534.3.1 class_()

```
template<bool compression>
std::size_t smtrat::uf_impl::uf_data_wrapper< void, compression >::class_ ( ) const [inline],
[noexcept]
```

15.534.3.2 increment_rank() template<bool compression>

```
void smtrat::uf_impl::uf_data_wrapper< void, compression >::increment_rank ( ) [inline], [noexcept]
```

15.534.3.3 rank() template<bool compression>

```
std::size_t smtrat::uf_impl::uf_data_wrapper< void, compression >::rank ( ) const [inline],
[noexcept]
```

15.534.3.4 set_class() template<bool compression>

```
void smtrat::uf_impl::uf_data_wrapper< void, compression >::set_class (
            std::size_t newclass ) [inline], [noexcept]
```

15.534.3.5 set_rank() template<bool compression>

```
void smtrat::uf_impl::uf_data_wrapper< void, compression >::set_rank (
            std::size_t rank_ ) [inline], [noexcept]
```

15.535 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)

15.535.1 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.

15.535.2 Constructor & Destructor Documentation

15.535.2.1 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]
```

15.535.3 Member Function Documentation

15.535.3.1 class_() template<bool compression>

```
std::size_t smtrat::uf_impl::uf_rank_and_class< compression >::class_ ( ) const [inline], [noexcept]
```

15.535.3.2 increment_rank() template<bool compression>

```
void smtrat::uf_impl::uf_rank_and_class< compression >::increment_rank ( ) [inline], [noexcept]
```

15.535.3.3 rank() template<bool compression>

```
std::size_t smtrat::uf_impl::uf_rank_and_class< compression >::rank ( ) const [inline], [noexcept]
```

15.535.3.4 set_class() template<bool compression>

```
void smtrat::uf_impl::uf_rank_and_class< compression >::set_class (
            std::size_t newclass ) [inline], [noexcept]
```

15.535.3.5 set_rank() template<bool compression>

```
void smtrat::uf_impl::uf_rank_and_class< compression >::set_rank (
            std::size_t rank ) [inline]
```

15.536 smtrat::uf_impl::uf_rank_and_class< true > Class Template 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)

15.536.1 Detailed Description

template<>
class smtrat::uf_impl::uf_rank_and_class< true >

Compressed implementation of uf_rank_and_class.

15.536.2 Constructor & Destructor Documentation

15.536.2.1 uf_rank_and_class() smtrat::uf_impl::uf_rank_and_class< true >::uf_rank_and_class (

```
            std::size_t class_index ) [inline]
```

15.536.3 Member Function Documentation

15.536.3.1 class_() std::size_t smtrat::uf_impl::uf_rank_and_class< true >::class_ ( ) const

```
[inline], [noexcept]
```

15.536.3.2 increment_rank() `void smtrat::uf_impl::uf_rank_and_class< true >::increment_rank ( ) [inline], [noexcept]`

15.536.3.3 rank() `std::size_t smtrat::uf_impl::uf_rank_and_class< true >::rank ( ) const [inline], [noexcept]`

15.536.3.4 set_class() `void smtrat::uf_impl::uf_rank_and_class< true >::set_class ( std::size_t newclass ) [inline], [noexcept]`

15.536.3.5 set_rank() `void smtrat::uf_impl::uf_rank_and_class< true >::set_rank ( std::size_t rank ) [inline]`

15.537 smtrat::UFCegarModule< Settings > Class Template Reference

`#include <UFCegarModule.h>`

Public Types

- typedef Settings SettingsType
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::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 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 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 _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)

  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 &_branching↩ Value) 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 _preferLeft↩ Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩ ::vector< FormulaT >())
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩ AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩ CaseWeak=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.

15.537.1 Member Typedef Documentation

15.537.1.1 SettingsType `template<typename Settings >`

```
typedef Settings smtrat::UFCegarModule< Settings >::SettingsType
```

15.537.2 Member Enumeration Documentation

15.537.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

15.537.3 Constructor & Destructor Documentation

15.537.3.1 UFCegarModule() `template<class Settings >`

```
smtrat::UFCegarModule< Settings >::UFCegarModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

15.537.3.2 ~UFCegarModule() `template<class Settings >`

```
smtrat::UFCegarModule< Settings >::~UFCegarModule ( )
```

15.537.4 Member Function Documentation

15.537.4.1 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.

15.537.4.2 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 >.

15.537.4.3 addCore()

```
template<class 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.

15.537.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.537.4.5 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.

15.537.4.6 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.

15.537.4.7 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.

15.537.4.8 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.

15.537.4.9 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.

15.537.4.10 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

_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.

15.537.4.11 allModels() `const std::vector<Model>& smtrat::Module::allModels ( ) const [inline], [inherited]`

Returns All satisfying assignments, if existent.

15.537.4.12 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.

15.537.4.13 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.

15.537.4.14 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.

15.537.4.15 branchAt() [1/6]

```
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]
```

15.537.4.16 branchAt() [2/6]

```
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]
```

15.537.4.17 branchAt() [3/6]

```
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]
```

15.537.4.18 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.537.4.19 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P
>> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            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]
```

15.537.4.20 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false )  [inline], [protected], [inherited]
```

15.537.4.21 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in smtrat::PModule.

15.537.4.22 checkCore()

```
template<class 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.

15.537.4.23 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
            std::vector< FormulaSetT >::const_iterator _infsubset,
            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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.537.4.24 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.

15.537.4.25 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.537.4.26 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.537.4.27 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.537.4.28 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.537.4.29 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.537.4.30 collectOrigins() [1/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

15.537.4.31 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

_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.

15.537.4.32 collectTheoryPropagations()

`void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.537.4.33 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ToInform ( ) const [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.537.4.34 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

15.537.4.35 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)

15.537.4.36 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

_constraint	The constraint to remove from internal data structures.

15.537.4.37 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.537.4.38 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.537.4.39 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.537.4.40 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.537.4.41 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.537.4.42 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.

15.537.4.43 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.

15.537.4.44 freeSplittingVariable()

```
static void smtrat::Module::freeSplittingVariable (
    const FormulaT & _splittingVariable ) [inline], [static], [inherited]
```

15.537.4.45 generateTrivialInfeasibleSubset()

```
void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]
```

Stores the trivial infeasible subset being the set of received formulas.

15.537.4.46 getBackendsAllModels()

```
void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]
```

Stores all models of a backend in the list of all models of this module.

15.537.4.47 getBackendsModel()

```
void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]
```

15.537.4.48 getInfeasibleSubsets() [1/2]

```
void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]
```

Copies the infeasible subsets of the passed formula.

15.537.4.49 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.537.4.50 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.537.4.51 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.

15.537.4.52 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.537.4.53 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.537.4.54 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.537.4.55 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 reveived formula, if the Boolean is true.

Reimplemented in smtrat::PModule.

15.537.4.56 hasLemmas()

`bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

15.537.4.57 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).

15.537.4.58 id()

`std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns A unique ID to identify this module instance.

15.537.4.59 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.

15.537.4.60 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.537.4.61 informBackends()

```
void smtrat::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.

15.537.4.62 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, and smtrat::ICPModule< ICPSettings1 >.

15.537.4.63 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.537.4.64 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::FPPModule< Settings >.

15.537.4.65 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.537.4.66 isLemmaLevel() `bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.537.4.67 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().

15.537.4.68 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.

15.537.4.69 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.537.4.70 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.537.4.71 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

_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.

15.537.4.72 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.

15.537.4.73 objective() `carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

15.537.4.74 originInReceivedFormula() `bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

15.537.4.75 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula. TODO: disable this method

15.537.4.76 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula.

15.537.4.77 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to the backends of this module.

15.537.4.78 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to this module.

15.537.4.79 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.537.4.80 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

_out	The stream to print the assignment on.

15.537.4.81 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const  [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.537.4.82 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

_out	The stream to print the assignment on.

15.537.4.83 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const  [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.537.4.84 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const  [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.537.4.85 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.

15.537.4.86 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

15.537.4.87 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩
InfeasibleSubset (
            ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

15.537.4.88 receivedVariable()

```
bool smtrat::Module::receivedVariable (
            carl::Variable::Arg _var ) const [inline], [inherited]
```

15.537.4.89 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

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

15.537.4.90 removeCore()

```
template<class 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

_subformula	The position of the subformula to remove.

Reimplemented from smtrat::Module.

15.537.4.91 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.537.4.92 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]
```

15.537.4.93 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.537.4.94 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.537.4.95 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]
```

Reimplemented in smtrat::PModule.

15.537.4.96 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

_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 in smtrat::PModule.

15.537.4.97 setId()

```
void smtrat::Module::setId ( std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.537.4.98 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.

15.537.4.99 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.

15.537.4.100 splitUnequalConstraint()

```
void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]
```

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.537.4.101 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.537.4.102 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 >.

15.537.4.103 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.537.4.104 updateModel() `template<class 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.

15.537.4.105 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).

15.537.5 Field Documentation

15.537.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.537.5.2 mAllModels

`std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.537.5.3 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.

15.537.5.4 mConstraintsToInform

`carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.537.5.5 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

15.537.5.6 mFirstPosInLastBranches

`std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.537.5.7 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.

15.537.5.8 mFirstUncheckedReceivedSubformula

`ModuleInput::const_iterator smtrat::Module::m↩FirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

15.537.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.537.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.537.5.11 mInfeasibleSubsets `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.537.5.12 mInformedConstraints `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.537.5.13 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.

15.537.5.14 mLemmas `std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.537.5.15 mModel `Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.537.5.16 mModelComputed `bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.537.5.17 mNumOfBranchVarsToStore `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.537.5.18 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.

15.537.5.19 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplitting↩Variables [static], [inherited]`

Reusable splitting variables.

15.537.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.537.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.537.5.22 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.

15.537.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

15.537.5.24 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).

15.537.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.538 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

15.538.1 Member Typedef Documentation

15.538.1.1 UEquality `typedef carl::UEquality smtrat::UFRewriter::UEquality`

15.538.1.2 UFInstance `typedef carl::UFInstance smtrat::UFRewriter::UFInstance`

15.538.1.3 UFunction `typedef carl::UninterpretedFunction smtrat::UFRewriter::UFunction`

15.538.1.4 UTerm `typedef carl::UTerm smtrat::UFRewriter::UTerm`

15.538.1.5 UVariable `typedef carl::UVariable smtrat::UFRewriter::UVariable`

15.538.2 Member Function Documentation

15.538.2.1 congruences() `const FormulaSetT& smtrat::UFRewriter::congruences ( ) const [inline], [noexcept]`

15.538.2.2 rewrite_ueq() `FormulaT smtrat::UFRewriter::rewrite_ueq ( const FormulaT & formula ) [inline]`

15.538.2.3 UFItoVar() `const std::unordered_map<UFInstance, UVariable>& smtrat::UFRewriter::UFItoVar ( ) const [inline], [noexcept]`

15.539 smtrat::expression::UnaryExpression Struct Reference

`#include <ExpressionContent.h>`

Public Member Functions

- UnaryExpression (UnaryType _type, Expression && _expression)

Data Fields

- UnaryType type
- Expression expression

15.539.1 Constructor & Destructor Documentation

15.539.1.1 UnaryExpression() `smtrat::expression::UnaryExpression::UnaryExpression ( UnaryType _type, Expression && _expression ) [inline]`

15.539.2 Field Documentation

15.539.2.1 expression `Expression smtrat::expression::UnaryExpression::expression`

15.539.2.2 type `UnaryType smtrat::expression::UnaryExpression::type`

15.540 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< boost::optional< std::string > > steps

15.540.1 Member Function Documentation

15.540.1.1 add()

```
void smtrat::cad::debug::UnifiedData::add (
    std::size_t step,
    const std::string & data ) [inline]
```

15.540.1.2 data()

```
const std::string& smtrat::cad::debug::UnifiedData::data (
    std::size_t step ) const [inline]
```

15.540.1.3 showsOn()

```
bool smtrat::cad::debug::UnifiedData::showsOn (
    std::size_t step ) const [inline]
```

15.540.2 Field Documentation

15.540.2.1 steps

```
std::vector<boost::optional<std::string> > smtrat::cad::debug::UnifiedData↩
::steps
```

15.541 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

15.541.1 Detailed Description

Types of the theory of equalities and uninterpreted functions.

15.541.2 Member Typedef Documentation

15.541.2.1 ConstTypes

```
typedef mpl::vector<carl::UTerm> smtrat::parser::types::UninterpretedTheory::ConstType
```

15.541.2.2 ExpressionTypes

```
typedef mpl::vector<carl::UTerm> smtrat::parser::types::UninterpretedTheory::Expr
```

15.541.2.3 TermType

```
typedef carl::mpl_variant_of<TermTypes>::type smtrat::parser::types::UninterpretedTheory:
```

15.541.2.4 TermTypes `typedef mpl::vector<carl::UTerm, carl::UVariable>` smtrat::parser::types::UninterpretedT

15.541.2.5 VariableTypes `typedef mpl::vector<carl::UVariable>` smtrat::parser::types::UninterpretedTheory::Var

15.542 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

15.542.1 Detailed Description

Implements the theory of equalities and uninterpreted functions.

15.542.2 Constructor & Destructor Documentation

15.542.2.1 UninterpretedTheory()

```
smtrat::parser::UninterpretedTheory::UninterpretedTheory (
    ParserState * state )
```

15.542.3 Member Function Documentation

15.542.3.1 addConstants()

```
static void smtrat::parser::AbstractTheory::addConstants (
    qi::symbols< char, types::ConstType > & ) [inline], [static], [inherited]
```

Initialize the global symbol table for constants.

15.542.3.2 addSimpleSorts()

```
static void smtrat::parser::AbstractTheory::addSimpleSorts (
    qi::symbols< char, carl::Sort > & ) [inline], [static], [inherited]
```

Initialize the global symbol table for simple sorts.

15.542.3.3 coupleBooleanVariables()

```
FormulaT smtrat::parser::UninterpretedTheory::coupleBooleanVariables (
    carl::Variable v,
    carl::UVariable uv ) [inline]
```

15.542.3.4 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.

15.542.3.5 expandDistinct()

```
template<typename T , typename Builder >
FormulaT smtrat::parser::AbstractTheory::expandDistinct (
    const std::vector< T > & values,
    const Builder & neqBuilder ) [inline], [inherited]
```

15.542.3.6 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.

15.542.3.7 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.

15.542.3.8 handleFunctionInstantiation()

```
bool smtrat::parser::UninterpretedTheory::handleFunction↩
Instantiation (
    const carl::UninterpretedFunction & f,
    const std::vector< types::TermType > & arguments,
    types::TermType & result,
    TheoryError & errors )
```

15.542.3.9 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.

15.542.3.10 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.

15.542.3.11 refreshVariable()

```
virtual bool smtrat::parser::AbstractTheory::refreshVariable (
    const types::VariableType & ,
    types::VariableType & ,
    TheoryError & errors )  [inline], [virtual], [inherited]
```

Reimplemented in smtrat::parser::BitvectorTheory.

15.542.3.12 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.

15.542.4 Field Documentation

15.542.4.1 mBoolSort

```
carl::Sort smtrat::parser::UninterpretedTheory::mBoolSort
```

15.542.4.2 mFalse

```
carl::UVariable smtrat::parser::UninterpretedTheory::mFalse
```

15.542.4.3 mInstantiatedArguments

```
std::map<types::TermType, carl::UTerm> smtrat::parser::↩
UninterpretedTheory::mInstantiatedArguments
```

15.542.4.4 mTrue

```
carl::UVariable smtrat::parser::UninterpretedTheory::mTrue
```

15.542.4.5 state

```
ParserState* smtrat::parser::AbstractTheory::state [inherited]
```

15.543 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 (Input↩ Iterator begin_, InputIterator end_)

  Initialize the union-find datastructure with an array of entries copied from the sequence [begin_,end_).

15.543.1 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 ceil(log2(#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(log2(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.

15.543.2 Constructor & Destructor Documentation

15.543.2.1 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.

15.543.2.2 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.

15.543.2.3 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.

15.543.2.4 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]
```

15.543.3 Member Function Documentation

15.543.3.1 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).

15.543.3.2 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).

15.543.3.3 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.

15.543.3.4 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.

15.543.3.5 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_).

15.543.3.6 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.

15.543.3.7 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.

15.543.3.8 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]
```

15.543.3.9 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.

15.543.3.10 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).

15.543.3.11 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.

15.543.3.12 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]
```

15.543.3.13 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.

15.543.3.14 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.

15.544 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

15.544.1 Member Typedef Documentation

15.544.1.1 Representative

```
template<typename T , typename Implementation >
using smtrat::UnionFindInterface< T, Implementation >::Representative = typename Implementation↩
::Representative
```

15.544.1.2 TranslateMap

```
template<typename T , typename Implementation >
using smtrat::UnionFindInterface< T, Implementation >::TranslateMap = std::unordered_map<T,
Value>
```

15.544.1.3 Value

```
template<typename T , typename Implementation >
using smtrat::UnionFindInterface< T, Implementation >::Value = typename Implementation::Value
```

15.544.2 Constructor & Destructor Documentation

15.544.2.1 UnionFindInterface()

```
template<typename T , typename Implementation >
smtrat::UnionFindInterface< T, Implementation >::UnionFindInterface (
            TranslateMap & translate )  [inline]
```

15.544.3 Member Function Documentation

15.544.3.1 backtrack()

```
template<typename T , typename Implementation >
void smtrat::UnionFindInterface< T, Implementation >::backtrack (
            T const & a,
            T const & b )  [inline], [noexcept]
```

15.544.3.2 find()

```
template<typename T , typename Implementation >
auto smtrat::UnionFindInterface< T, Implementation >::find (
            T const & val ) -> Representative   [inline], [noexcept]
```

15.544.3.3 has_variable()

```
template<typename T , typename Implementation >
bool smtrat::UnionFindInterface< T, Implementation >::has_variable (
            T const & var )  [inline], [noexcept]
```

15.544.3.4 init()

```
template<typename T , typename Implementation >
template<typename Container >
void smtrat::UnionFindInterface< T, Implementation >::init (
            Container const & cont )  [inline], [noexcept]
```

15.544.3.5 insert_value()

```
template<typename T , typename Implementation >
Value smtrat::UnionFindInterface< T, Implementation >::insert_value (
            T const & var )  [inline], [noexcept]
```

15.544.3.6 merge()

```
template<typename T , typename Implementation >
void smtrat::UnionFindInterface< T, Implementation >::merge (
            T const & a,
            T const & b )  [inline], [noexcept]
```

15.545 smtrat::UnionFindModule< Settings > Class Template Reference

`#include <UnionFindModule.h>`

Public Types

- typedef Settings SettingsType
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::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 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 _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 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 &_branching↩Value) 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 _preferLeft↩Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩::vector< FormulaT >())

- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩CaseWeak=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.

15.545.1 Member Typedef Documentation

15.545.1.1 SettingsType `template<typename Settings >`

```
typedef Settings smtrat::UnionFindModule< Settings >::SettingsType
```

15.545.2 Member Enumeration Documentation

15.545.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

15.545.3 Constructor & Destructor Documentation

15.545.3.1 UnionFindModule() `template<class Settings >`

```
smtrat::UnionFindModule< Settings >::UnionFindModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

15.545.3.2 ~UnionFindModule() `template<class Settings >`

```
smtrat::UnionFindModule< Settings >::~UnionFindModule ( )
```

15.545.4 Member Function Documentation

15.545.4.1 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.

15.545.4.2 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 >.

15.545.4.3 addCore()

```
template<class 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

_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.

15.545.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.545.4.5 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.

15.545.4.6 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.

15.545.4.7 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.

15.545.4.8 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.

15.545.4.9 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.

15.545.4.10 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

_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.

15.545.4.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const  [inline],
[inherited]
```

Returns All satisfying assignments, if existent.

15.545.4.12 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.

15.545.4.13 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.

15.545.4.14 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.

15.545.4.15 branchAt() [1/6]

```
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]
```

15.545.4.16 branchAt() [2/6]

```
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]
```

15.545.4.17 branchAt() [3/6]

```
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]
```

15.545.4.18 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.545.4.19 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P
>> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            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]
```

15.545.4.20 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P
>> = carl::dummy>
bool smtrat::Module::branchAt (
            const typename P::PolyType & _poly,
            bool _integral,
            const Rational & _value,
            std::vector< FormulaT > && _premise,
            bool _leftCaseWeak = true,
            bool _preferLeftCase = true,
            bool _useReceivedFormulaAsPremise = false )  [inline], [protected], [inherited]
```

15.545.4.21 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in smtrat::PModule.

15.545.4.22 checkCore()

```
template<class 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 smtrat::Module.

15.545.4.23 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
    std::vector< FormulaSetT >::const_iterator _infsubset,
    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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.545.4.24 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.

15.545.4.25 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.545.4.26 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.545.4.27 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.545.4.28 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.545.4.29 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.545.4.30 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

15.545.4.31 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

_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.

15.545.4.32 collectTheoryPropagations()

`void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.545.4.33 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.545.4.34 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

15.545.4.35 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)

15.545.4.36 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

_constraint	The constraint to remove from internal data structures.

15.545.4.37 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.545.4.38 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.545.4.39 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.545.4.40 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.545.4.41 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.545.4.42 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.

15.545.4.43 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.

15.545.4.44 freeSplittingVariable()

```
static void smtrat::Module::freeSplittingVariable (
    const FormulaT & _splittingVariable ) [inline], [static], [inherited]
```

15.545.4.45 generateTrivialInfeasibleSubset()

```
void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]
```

Stores the trivial infeasible subset being the set of received formulas.

15.545.4.46 getBackendsAllModels()

```
void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]
```

Stores all models of a backend in the list of all models of this module.

15.545.4.47 getBackendsModel()

```
void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]
```

15.545.4.48 getInfeasibleSubsets() [1/2]

```
void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]
```

Copies the infeasible subsets of the passed formula.

15.545.4.49 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.545.4.50 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.545.4.51 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.

15.545.4.52 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.545.4.53 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.545.4.54 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.545.4.55 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 reveived formula, if the Boolean is true.

Reimplemented in smtrat::PModule.

15.545.4.56 hasLemmas()

`bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

15.545.4.57 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).

15.545.4.58 id()

`std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns A unique ID to identify this module instance.

15.545.4.59 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.

15.545.4.60 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.545.4.61 informBackends()

```
void smtrat::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.

15.545.4.62 informCore()

```
template<class 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

_constraint	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.

15.545.4.63 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.545.4.64 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::FPPModule< Settings >.

15.545.4.65 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.545.4.66 isLemmaLevel() `bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.545.4.67 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().

15.545.4.68 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.

15.545.4.69 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.545.4.70 model()

`const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.545.4.71 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

_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.

15.545.4.72 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.

15.545.4.73 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

15.545.4.74 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
    const FormulaT & _origin ) const [protected], [inherited]
```

15.545.4.75 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula. TODO: disable this method

15.545.4.76 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula.

15.545.4.77 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to the backends of this module.

15.545.4.78 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to this module.

15.545.4.79 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.545.4.80 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

_out	The stream to print the assignment on.

15.545.4.81 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const  [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.545.4.82 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

_out	The stream to print the assignment on.

15.545.4.83 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const  [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.545.4.84 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const  [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.545.4.85 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.

15.545.4.86 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

15.545.4.87 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩InfeasibleSubset (
            ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

15.545.4.88 receivedVariable()

```
bool smtrat::Module::receivedVariable (
            carl::Variable::Arg _var ) const [inline], [inherited]
```

15.545.4.89 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

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

15.545.4.90 removeCore()

```
template<class 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

_subformula	The position of the subformula to remove.

Reimplemented from smtrat::Module.

15.545.4.91 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.545.4.92 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]
```

15.545.4.93 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.545.4.94 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.545.4.95 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]
```

Reimplemented in smtrat::PModule.

15.545.4.96 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

_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 in smtrat::PModule.

15.545.4.97 setId()

```
void smtrat::Module::setId ( std::size_t _id ) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.545.4.98 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.

15.545.4.99 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.

15.545.4.100 splitUnequalConstraint()

```
void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]
```

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.545.4.101 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.545.4.102 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 >.

15.545.4.103 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.545.4.104 updateModel() `template<class 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.

15.545.4.105 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).

15.545.5 Field Documentation

15.545.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.545.5.2 mAllModels

`std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.545.5.3 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.

15.545.5.4 mConstraintsToInform

`carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.545.5.5 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

15.545.5.6 mFirstPosInLastBranches

`std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.545.5.7 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.

15.545.5.8 mFirstUncheckedReceivedSubformula

`ModuleInput::const_iterator smtrat::Module::m↩ FirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

15.545.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.545.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.545.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.545.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.545.5.13 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.

15.545.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.545.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.545.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.545.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.545.5.18 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.

15.545.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplitting↩Variables [static], [inherited]`

Reusable splitting variables.

15.545.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.545.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.545.5.22 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.

15.545.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

15.545.5.24 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).

15.545.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.546 smtrat::SATModule< Settings >::UnorderedClauseLookup::UnorderedClauseHasher Struct Reference

`#include <SATModule.h>`

Public Member Functions

- std::size_t operator() (const std::vector< Minisat::Lit > &cl) const

15.546.1 Member Function Documentation

15.546.1.1 operator()() `template<class Settings > std::size_t smtrat::SATModule< Settings >::UnorderedClauseLookup::UnorderedClauseHasher↩ ::operator() ( const std::vector< Minisat::Lit > & cl ) const [inline]`

15.547 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 ()

15.547.1 Constructor & Destructor Documentation

15.547.1.1 UnsatCore()

```
template<typename Solver, UnsatCoreStrategy Strategy>
smtrat::UnsatCore< Solver, Strategy >::UnsatCore (
            Solver & s )  [inline]
```

15.547.2 Member Function Documentation

15.547.2.1 active()

```
template<typename Solver, UnsatCoreStrategy Strategy>
bool smtrat::UnsatCore< Solver, Strategy >::active ( ) const  [inline]
```

15.547.2.2 add_annotated_name()

```
template<typename Solver, UnsatCoreStrategy Strategy>
void smtrat::UnsatCore< Solver, Strategy >::add_annotated_name (
            const FormulaT & formula,
            const std::string & name )  [inline]
```

15.547.2.3 compute()

```
template<typename Solver, UnsatCoreStrategy Strategy>
std::pair<Answer,std::vector<std::string> > smtrat::UnsatCore< Solver, Strategy >::compute (
)  [inline]
```

15.547.2.4 compute_core()

```
template<typename Solver, UnsatCoreStrategy Strategy>
std::pair<Answer,FormulasT> smtrat::UnsatCore< Solver, Strategy >::compute_core (
            const FormulasT & formulas )  [inline]
```

15.547.2.5 remove_annotated_name()

```
template<typename Solver, UnsatCoreStrategy Strategy>
void smtrat::UnsatCore< Solver, Strategy >::remove_annotated_name (
            const FormulaT & formula )  [inline]
```

15.547.2.6 reset()

```
template<typename Solver, UnsatCoreStrategy Strategy>
void smtrat::UnsatCore< Solver, Strategy >::reset ( )  [inline]
```

15.548 smtrat::unsatcore::UnsatCoreBackend< Solver, Strategy > Class Template Reference

```
#include <UnsatCore.h>
```

15.549 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 ()

15.549.1 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 <=> f_i)
- Find a satisfying assignment (which will set some f_i to false)
-

15.549.2 Constructor & Destructor Documentation

15.549.2.1 UnsatCoreBackend()

```
template<typename Solver >
smtrat::unsatcore::UnsatCoreBackend< Solver, UnsatCoreStrategy::ModelExclusion >::UnsatCoreBackend
(
            Solver & s,
            const FormulasT & fs )  [inline]
```

15.549.3 Member Function Documentation

15.549.3.1 compute()

```
template<typename Solver >
Answer smtrat::unsatcore::UnsatCoreBackend< Solver, UnsatCoreStrategy::ModelExclusion >↩
::compute ( )  [inline]
```

15.549.3.2 processAssignment()

```
template<typename Solver >
void smtrat::unsatcore::UnsatCoreBackend< Solver, UnsatCoreStrategy::ModelExclusion >::process↩
Assignment ( )  [inline]
```

15.549.3.3 run()

```
template<typename Solver >
std::pair<Answer, FormulasT> smtrat::unsatcore::UnsatCoreBackend< Solver, UnsatCoreStrategy::ModelExclusion
>::run ( )  [inline]
```

15.550 smtrat::vs::unsignedTripleCmp Struct Reference

```
#include <State.h>
```

Public Member Functions

- bool operator() (UnsignedTriple n1, UnsignedTriple n2) const

15.550.1 Member Function Documentation

15.550.1.1 operator()()

```
bool smtrat::vs::unsignedTripleCmp::operator() (
    UnsignedTriple n1,
    UnsignedTriple n2 ) const [inline]
```

15.551 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

15.551.1 Constructor & Destructor Documentation

15.551.1.1 UserFunctionInstantiator()

```
smtrat::parser::UserFunctionInstantiator::UserFunction↩
Instantiator (
    const std::vector< types::VariableType > & arguments,
    const carl::Sort & sort,
    const types::TermType & definition,
    const std::set< types::VariableType > & auxiliaries ) [inline]
```

15.551.2 Member Function Documentation

15.551.2.1 convert() [1/2]

```
template<typename T >
bool smtrat::parser::FunctionInstantiator::convert (
    const std::vector< types::TermType > & from,
    std::vector< T > & to ) const [inline], [inherited]
```

15.551.2.2 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]
```

15.551.2.3 operator()()

```
virtual bool smtrat::parser::FunctionInstantiator::operator() (
    const std::vector< types::TermType > & ,
    types::TermType & ,
    TheoryError & errors ) const [inline], [virtual], [inherited]
```

15.551.3 Field Documentation

15.551.3.1 arguments `std::vector<types::VariableType> smtrat::parser::UserFunctionInstantiator↩::arguments`

15.551.3.2 auxiliaries `std::set<types::VariableType> smtrat::parser::UserFunctionInstantiator↩::auxiliaries`

15.551.3.3 definition `types::TermType smtrat::parser::UserFunctionInstantiator::definition`

15.551.3.4 globalFormulas `FormulasT smtrat::parser::UserFunctionInstantiator::globalFormulas`

15.551.3.5 sort `carl::Sort smtrat::parser::UserFunctionInstantiator::sort`

15.552 smtrat::validation::ValidationCollector Class Reference

`#include <ValidationCollector.h>`

Public Member Functions

- ValidationPoint & get (const std::string &channel, const std::string &name)
- const auto & points () const

15.552.1 Member Function Documentation

15.552.1.1 get()

```
ValidationPoint& smtrat::validation::ValidationCollector::get (
    const std::string & channel,
    const std::string & name ) [inline]
```

15.552.1.2 points()

```
const auto& smtrat::validation::ValidationCollector::points ( ) const [inline]
```

15.553 smtrat::validation::ValidationPoint Class Reference

`#include <ValidationPoint.h>`

Public Member Functions

- void set_identifier (const std::string &channel, const std::string &name)
- const auto & channel () const
- const auto & name () const
- auto identifier () const
- const auto & formulas () const
- std::size_t add (const FormulaT &f, bool consistent)
- std::size_t add (const FormulasT &fs, bool consistent)
- std::size_t add (const FormulaSetT &fss, bool consistent)
- std::size_t add (const ConstraintT &c, bool consistent)
- std::size_t add (const ConstraintsT &cs, bool consistent)

15.553.1 Member Function Documentation

15.553.1.1 add() [1/5]

```
std::size_t smtrat::validation::ValidationPoint::add (
    const ConstraintsT & cs,
    bool consistent ) [inline]
```

15.553.1.2 add() [2/5]

```
std::size_t smtrat::validation::ValidationPoint::add (
    const ConstraintT & c,
    bool consistent ) [inline]
```

15.553.1.3 add() [3/5]

```
std::size_t smtrat::validation::ValidationPoint::add (
    const FormulaSetT & fss,
    bool consistent ) [inline]
```

15.553.1.4 add() [4/5]

```
std::size_t smtrat::validation::ValidationPoint::add (
    const FormulasT & fs,
    bool consistent ) [inline]
```

15.553.1.5 add() [5/5]

```
std::size_t smtrat::validation::ValidationPoint::add (
    const FormulaT & f,
    bool consistent ) [inline]
```

15.553.1.6 channel()

```
const auto& smtrat::validation::ValidationPoint::channel ( ) const [inline]
```

15.553.1.7 formulas()

```
const auto& smtrat::validation::ValidationPoint::formulas ( ) const [inline]
```

15.553.1.8 identifier()

```
auto smtrat::validation::ValidationPoint::identifier ( ) const [inline]
```

15.553.1.9 name()

```
const auto& smtrat::validation::ValidationPoint::name ( ) const [inline]
```

15.553.1.10 set_identifier()

```
void smtrat::validation::ValidationPoint::set_identifier (
    const std::string & channel,
    const std::string & name ) [inline]
```

15.554 smtrat::validation::ValidationPrinter< SOF > Struct Template Reference

```
#include <ValidationPrinter.h>
```

15.555 smtrat::validation::ValidationSettings Struct Reference

```
#include <ValidationSettings.h>
```

Public Member Functions
- bool channel_active (const std::string &key) const

Data Fields
- bool export_as_smtlib
- std::string smtlib_filename
- std::vector< std::string > channels

15.555.1 Member Function Documentation

15.555.1.1 channel_active()

```
bool smtrat::validation::ValidationSettings::channel_active (
    const std::string & key ) const [inline]
```

15.555.2 Field Documentation

15.555.2.1 channels

```
std::vector<std::string> smtrat::validation::ValidationSettings::channels
```

15.555.2.2 export_as_smtlib

```
bool smtrat::validation::ValidationSettings::export_as_smtlib
```

15.555.2.3 smtlib_filename

```
std::string smtrat::validation::ValidationSettings::smtlib_filename
```

15.556 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 Value< T > &_value)
- 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 isZero () const
- T sign () const

  Computes the sign: 0 if main & delta part are unequal 0.
- void print (std::ostream &_out=std::cout) const

15.556.1 Constructor & Destructor Documentation

15.556.1.1 Value() [1/4] template<class T>

```
smtrat::lra::Value< T >::Value ( )
```

15.556.1.2 Value() [2/4] template<class T>

```
smtrat::lra::Value< T >::Value (
            const T & _num )
```

Parameters

_num	

15.556.1.3 Value() [3/4] template<class T>

```
smtrat::lra::Value< T >::Value (
            const T & _num1,
            const T & _num2 )
```

Parameters

_num1	
_num2	

15.556.1.4 Value() [4/4] template<class T>

```
smtrat::lra::Value< T >::Value (
            const Value< T > & _orig )
```

Parameters

_orig	

15.556.1.5 ~Value() template<class T>

```
virtual smtrat::lra::Value< T >::~Value ( ) [virtual]
```

15.556.2 Member Function Documentation

15.556.2.1 abs() template<class T>

```
Value<T> smtrat::lra::Value< T >::abs ( ) const [inline]
```

15.556.2.2 abs_() template<class T>

```
const Value<T>& smtrat::lra::Value< T >::abs_ ( ) [inline]
```

15.556.2.3 deltaPart() template<class T>

```
const T& smtrat::lra::Value< T >::deltaPart ( ) const [inline]
```

Returns

15.556.2.4 isZero() template<class T>

```
bool smtrat::lra::Value< T >::isZero ( ) const [inline]
```

15.556.2.5 mainPart() template<class T>

```
const T& smtrat::lra::Value< T >::mainPart ( ) const [inline]
```

Returns

15.556.2.6 operator"!=() [1/2] template<class T>

```
bool smtrat::lra::Value< T >::operator!= (
            const T & _a ) const [inline]
```

15.556.2.7 operator"!=() [2/2] template<class T>

```
bool smtrat::lra::Value< T >::operator!= (
            const Value< T > & _value ) const [inline]
```

15.556.2.8 operator∗() [1/2] template<class T>

```
Value<T> smtrat::lra::Value< T >::operator* (
            const T & _a ) const
```

Parameters

↩ _↩ *a*	

Returns

15.556.2.9 operator∗() [2/2] template<class T>

```
Value<T> smtrat::lra::Value< T >::operator* (
            const Value< T > & _value ) const
```

Parameters

_value	

15.556.2.10 operator∗=() [1/2] template<class T>

```
void smtrat::lra::Value< T >::operator*= (
            const T & _a )
```

Parameters

↩ ↩ *a*	

15.556.2.11 operator∗=() [2/2] template<class T>

```
void smtrat::lra::Value< T >::operator*= (
            const Value< T > & _value )
```

Parameters

_value	

15.556.2.12 operator+() template<class T>

```
Value<T> smtrat::lra::Value< T >::operator+ (
            const Value< T > & _value ) const
```

Parameters

_value	

Returns

15.556.2.13 operator+=() template<class T>

```
void smtrat::lra::Value< T >::operator+= (
            const Value< T > & _value )
```

Parameters

_value	

15.556.2.14 operator-() template<class T>

```
Value<T> smtrat::lra::Value< T >::operator- (
            const Value< T > & _value ) const
```

Parameters

_value	

Returns

15.556.2.15 operator-=() template<class T>

```
void smtrat::lra::Value< T >::operator-= (
            const Value< T > & _value )
```

Parameters

_value	

15.556.2.16 operator/() template<class T>

```
Value<T> smtrat::lra::Value< T >::operator/ (
            const T & _a ) const
```

Parameters

↩	
_↩	
a	

Returns

15.556.2.17 operator/=() template<class T>

```
void smtrat::lra::Value< T >::operator/= (
            const T & _a )
```

Parameters

↩	
_↩	
a	

15.556.2.18 operator<() [1/2] template<class T>

```
bool smtrat::lra::Value< T >::operator< (
            const T & _a ) const
```

Parameters

↩ _↩ *a*	

Returns

15.556.2.19 operator<() [2/2] template<class T>

```
bool smtrat::lra::Value< T >::operator< (
            const Value< T > & _value ) const
```

Parameters

_value	

Returns

15.556.2.20 operator<=() [1/2] template<class T>

```
bool smtrat::lra::Value< T >::operator<= (
            const T & _a ) const
```

Parameters

↩ _↩ *a*	

Returns

15.556.2.21 operator<=() [2/2] template<class T>

```
bool smtrat::lra::Value< T >::operator<= (
            const Value< T > & _value ) const
```

Parameters

_value	

Returns

15.556.2.22 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.

15.556.2.23 operator==() [1/2] template<class T>

```
bool smtrat::lra::Value< T >::operator== (
            const T & _a ) const
```

Parameters

↩	
_↩	
a	

Returns

15.556.2.24 operator==() [2/2] template<class T>

```
bool smtrat::lra::Value< T >::operator== (
            const Value< T > & _value ) const
```

Parameters

_value	

Returns

15.556.2.25 operator>() [1/2] template<class T>

```
bool smtrat::lra::Value< T >::operator> (
            const T & _a ) const
```

Parameters

↩	
_↩	
a	

Returns

15.556.2.26 operator>() [2/2] template<class T>

```
bool smtrat::lra::Value< T >::operator> (
            const Value< T > & _value ) const  [inline]
```

15.556.2.27 operator>=() [1/2] template<class T>

```
bool smtrat::lra::Value< T >::operator>= (
            const T & _a ) const
```

Parameters

↩ _↩ *a*	

Returns

15.556.2.28 operator>=() [2/2] template<class T>

```
bool smtrat::lra::Value< T >::operator>= (
            const Value< T > & _value ) const  [inline]
```

15.556.2.29 print() template<class T>

```
void smtrat::lra::Value< T >::print (
            std::ostream & _out = std::cout ) const
```

Parameters

_out	

15.556.2.30 sign() template<class T>

```
T smtrat::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.

15.556.2.31 toString() template<class T>

```
const std::string smtrat::lra::Value< T >::toString ( ) const
```

Returns

15.557 smtrat::lra::Variable< T1, T2 > Class Template Reference

```
#include <Bound.h>
```

Public Member Functions

- Variable (size_t _position, const typename Poly::PolyType *_expression, ModuleInput::iterator _defaultBound↩ Position, bool _isInteger, size_t _id)
- Variable (typename std::list< std::list< std::pair< Variable< T1, T2 > *, T2 >>>::iterator _positionIn↩ NonActives, 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 isInteger () 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 getId () 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 EvalRationalMap &_ass) const
- FormulaT inConflictWith (const EvalRationalMap &_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

15.557.1 Constructor & Destructor Documentation

15.557.1.1 Variable() [1/2]

```
template<typename T1, typename T2>
smtrat::lra::Variable< T1, T2 >::Variable (
            size_t _position,
            const typename Poly::PolyType * _expression,
            ModuleInput::iterator _defaultBoundPosition,
            bool _isInteger,
            size_t _id )
```

Parameters

_position	
_expression	
_defaultBoundPosition	
_isInteger	
_id	

15.557.1.2 Variable() [2/2]

```
template<typename T1, typename T2>
smtrat::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

_positionInNonActives	
_expression	
_defaultBoundPosition	
_isInteger	
_id	

15.557.1.3 ~Variable()

```
template<typename T1, typename T2>
virtual smtrat::lra::Variable< T1, T2 >::~Variable ( )  [virtual]
```

15.557.2 Member Function Documentation

15.557.2.1 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

_val	
_position	
_constraint	

Returns

15.557.2.2 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

_val	
_position	
_constraint	
_deduced	

Returns

15.557.2.3 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

_val	
_position	
_constraint	
_deduced	

Returns

15.557.2.4 assignment()

```
template<typename T1, typename T2>
const Value<T1>& smtrat::lra::Variable< T1, T2 >::assignment ( ) const  [inline]
```

Returns

15.557.2.5 conflictActivity()

```
template<typename T1, typename T2>
double smtrat::lra::Variable< T1, T2 >::conflictActivity ( ) const  [inline]
```

Returns

15.557.2.6 deactivateBound()

```
template<typename T1, typename T2>
bool smtrat::lra::Variable< T1, T2 >::deactivateBound (
            const Bound< T1, T2 > * bound,
            ModuleInput::iterator _position )
```

Parameters

bound	
_position	

Returns

15.557.2.7 expression()

```
template<typename T1, typename T2>
const Poly::PolyType& smtrat::lra::Variable< T1, T2 >::expression ( ) const  [inline]
```

Returns

15.557.2.8 factor()

```
template<typename T1, typename T2>
const T2& smtrat::lra::Variable< T1, T2 >::factor ( ) const  [inline]
```

Returns

15.557.2.9 getDefiningOrigins()

```
template<typename T1, typename T2>
FormulasT smtrat::lra::Variable< T1, T2 >::getDefiningOrigins ( ) const
```

Returns

15.557.2.10 getId()

```
template<typename T1, typename T2>
size_t smtrat::lra::Variable< T1, T2 >::getId ( ) const  [inline]
```

15.557.2.11 getVariableBounds()

```
template<typename T1, typename T2>
RationalInterval smtrat::lra::Variable< T1, T2 >::getVariableBounds ( ) const
```

Returns

15.557.2.12 hasBound()

```
template<typename T1, typename T2>
bool smtrat::lra::Variable< T1, T2 >::hasBound ( ) const [inline]
```

Returns

15.557.2.13 inConflictWith()

```
template<typename T1, typename T2>
FormulaT smtrat::lra::Variable< T1, T2 >::inConflictWith (
            const EvalRationalMap & _ass ) const [inline]
```

Parameters

_ass	

Returns

15.557.2.14 infimum()

```
template<typename T1, typename T2>
const Bound<T1, T2>& smtrat::lra::Variable< T1, T2 >::infimum ( ) const [inline]
```

Returns

15.557.2.15 involvesEquation()

```
template<typename T1, typename T2>
bool smtrat::lra::Variable< T1, T2 >::involvesEquation ( ) const [inline]
```

Returns

15.557.2.16 isBasic()

```
template<typename T1, typename T2>
bool smtrat::lra::Variable< T1, T2 >::isBasic ( ) const [inline]
```

Returns

15.557.2.17 isConflicting()

```
template<typename T1, typename T2>
bool smtrat::lra::Variable< T1, T2 >::isConflicting ( ) const  [inline]
```

15.557.2.18 isInteger()

```
template<typename T1, typename T2>
bool smtrat::lra::Variable< T1, T2 >::isInteger ( ) const  [inline]
```

Returns

15.557.2.19 isOriginal()

```
template<typename T1, typename T2>
bool smtrat::lra::Variable< T1, T2 >::isOriginal ( ) const  [inline]
```

Returns

15.557.2.20 isSatisfiedBy()

```
template<typename T1, typename T2>
unsigned smtrat::lra::Variable< T1, T2 >::isSatisfiedBy (
            const EvalRationalMap & _ass ) const  [inline]
```

Parameters

_ass	

Returns

15.557.2.21 lastConsistentAssignment()

```
template<typename T1, typename T2>
const Value<T1>& smtrat::lra::Variable< T1, T2 >::lastConsistentAssignment ( ) const  [inline]
```

Returns

15.557.2.22 lowerbounds()

```
template<typename T1, typename T2>
const Bound<T1, T2>::BoundSet& smtrat::lra::Variable< T1, T2 >::lowerbounds ( ) const  [inline]
```

Returns

15.557.2.23 operator"!=()

```
template<typename T1, typename T2>
bool smtrat::lra::Variable< T1, T2 >::operator!= (
            const Variable< T1, T2 > & _variable ) const  [inline]
```

15.557.2.24 operator<()

```
template<typename T1, typename T2>
bool smtrat::lra::Variable< T1, T2 >::operator< (
            const Variable< T1, T2 > & _variable ) const  [inline]
```

Parameters

_bound	

Returns

15.557.2.25 operator==()

```
template<typename T1, typename T2>
bool smtrat::lra::Variable< T1, T2 >::operator== (
            const Variable< T1, T2 > & _variable ) const  [inline]
```

15.557.2.26 operator>()

```
template<typename T1, typename T2>
bool smtrat::lra::Variable< T1, T2 >::operator> (
            const Variable< T1, T2 > & _variable ) const  [inline]
```

15.557.2.27 pExpression()

```
template<typename T1, typename T2>
const Poly::PolyType* smtrat::lra::Variable< T1, T2 >::pExpression ( ) const  [inline]
```

Returns

15.557.2.28 pInfimum()

```
template<typename T1, typename T2>
const Bound<T1, T2>* smtrat::lra::Variable< T1, T2 >::pInfimum ( ) const  [inline]
```

Returns

15.557.2.29 position()

```
template<typename T1, typename T2>
size_t smtrat::lra::Variable< T1, T2 >::position ( ) const  [inline]
```

Returns

15.557.2.30 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

15.557.2.31 print()

```
template<typename T1, typename T2>
void smtrat::lra::Variable< T1, T2 >::print (
            std::ostream & _out = std::cout ) const
```

Parameters

_out	

15.557.2.32 printAllBounds()

```
template<typename T1, typename T2>
void smtrat::lra::Variable< T1, T2 >::printAllBounds (
            std::ostream & _out = std::cout,
            const std::string _init = "" ) const
```

Parameters

_out	
_init	

15.557.2.33 pSupremum()

```
template<typename T1, typename T2>
const Bound<T1, T2>* smtrat::lra::Variable< T1, T2 >::pSupremum ( ) const  [inline]
```

Returns

15.557.2.34 rAssignment()

```
template<typename T1, typename T2>
Value<T1>& smtrat::lra::Variable< T1, T2 >::rAssignment ( )  [inline]
```

Returns

15.557.2.35 removeBound()

```
template<typename T1, typename T2>
void smtrat::lra::Variable< T1, T2 >::removeBound (
            const Bound< T1, T2 > * _bound )
```

Parameters

_bound	

15.557.2.36 rFactor()

```
template<typename T1, typename T2>
T2& smtrat::lra::Variable< T1, T2 >::rFactor ( )  [inline]
```

Returns

15.557.2.37 rLowerbounds()

```
template<typename T1, typename T2>
Bound<T1, T2>::BoundSet& smtrat::lra::Variable< T1, T2 >::rLowerbounds ( ) [inline]
```

Returns

15.557.2.38 rLowerBoundsSize()

```
template<typename T1, typename T2>
size_t smtrat::lra::Variable< T1, T2 >::rLowerBoundsSize ( ) [inline]
```

Returns

15.557.2.39 rPosition()

```
template<typename T1, typename T2>
size_t& smtrat::lra::Variable< T1, T2 >::rPosition ( ) [inline]
```

Returns

15.557.2.40 rSize()

```
template<typename T1, typename T2>
size_t& smtrat::lra::Variable< T1, T2 >::rSize ( ) [inline]
```

Returns

15.557.2.41 rStartEntry()

```
template<typename T1, typename T2>
EntryID& smtrat::lra::Variable< T1, T2 >::rStartEntry ( ) [inline]
```

Returns

15.557.2.42 rUpperbounds()

```
template<typename T1, typename T2>
Bound<T1, T2>::BoundSet& smtrat::lra::Variable< T1, T2 >::rUpperbounds ( ) [inline]
```

Returns

15.557.2.43 rUpperBoundsSize()

```
template<typename T1, typename T2>
size_t smtrat::lra::Variable< T1, T2 >::rUpperBoundsSize ( ) [inline]
```

Returns

15.557.2.44 setBasic()

```
template<typename T1, typename T2>
void smtrat::lra::Variable< T1, T2 >::setBasic (
            bool _basic ) [inline]
```

Parameters

_basic	

15.557.2.45 setInfimum()

```
template<typename T1, typename T2>
void smtrat::lra::Variable< T1, T2 >::setInfimum (
            const Bound< T1, T2 > * _infimum )  [inline]
```

Parameters

_infimum	

15.557.2.46 setPosition()

```
template<typename T1, typename T2>
void smtrat::lra::Variable< T1, T2 >::setPosition (
            size_t _position )  [inline]
```

Parameters

_position	

15.557.2.47 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

_positionInNonActives	

15.557.2.48 setSupremum()

```
template<typename T1, typename T2>
void smtrat::lra::Variable< T1, T2 >::setSupremum (
            const Bound< T1, T2 > * _supremum )  [inline]
```

Parameters

_supremum	

15.557.2.49 size()

```
template<typename T1, typename T2>
size_t smtrat::lra::Variable< T1, T2 >::size ( ) const  [inline]
```

Returns

15.557.2.50 startEntry() `template<typename T1, typename T2>`

`EntryID smtrat::lra::Variable< T1, T2 >::startEntry ( ) const [inline]`

Returns

15.557.2.51 storeAssignment() `template<typename T1, typename T2>`

`void smtrat::lra::Variable< T1, T2 >::storeAssignment ( ) [inline]`

15.557.2.52 supremum() `template<typename T1, typename T2>`

`const Bound<T1, T2>& smtrat::lra::Variable< T1, T2 >::supremum ( ) const [inline]`

Returns

15.557.2.53 updateConflictActivity() `template<typename T1, typename T2>`

`void smtrat::lra::Variable< T1, T2 >::updateConflictActivity ( ) [inline]`

15.557.2.54 upperbounds() `template<typename T1, typename T2>`

`const Bound<T1, T2>::BoundSet& smtrat::lra::Variable< T1, T2 >::upperbounds ( ) const [inline]`

Returns

15.558 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 > * pInfimum () const
- const Bound< T > & infimum () const
- const BoundSet & upperbounds () const
- const BoundSet & lowerbounds () const

15.558.1 Detailed Description

template<typename T>
class smtrat::vb::Variable< T >

Class for a variable.

15.558.2 Constructor & Destructor Documentation

15.558.2.1 Variable()

```
template<typename T>
smtrat::vb::Variable< T >::Variable ( )
```

Constructs this variable.

15.558.2.2 ~Variable()

```
template<typename T>
smtrat::vb::Variable< T >::~Variable ( )
```

15.558.3 Member Function Documentation

15.558.3.1 addBound()

```
template<typename T>
const Bound<T>* smtrat::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

_constraint	A pointer to a constraint of the form ax~b.
_var	Hence, the variable x.
_origin	The origin of the constraint. This is could be the constraint itself or anything else in the data structure which uses this object.
_limit	

Returns The added bound.

15.558.3.2 conflicting()

```
template<typename T>
bool smtrat::vb::Variable< T >::conflicting ( ) const
```

Returns true, if there is a conflict; false, otherwise.

15.558.3.3 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.

15.558.3.4 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.

15.558.3.5 infimum()

```
template<typename T>
const Bound<T>& smtrat::vb::Variable< T >::infimum ( ) const [inline]
```

Returns A constant reference to the infimum of this variable.

15.558.3.6 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.

15.558.3.7 pInfimum()

```
template<typename T>
const Bound<T>* smtrat::vb::Variable< T >::pInfimum ( ) const [inline]
```

Returns A pointer to the infimum of this variable.

15.558.3.8 pSupremum()

```
template<typename T>
const Bound<T>* smtrat::vb::Variable< T >::pSupremum ( ) const [inline]
```

Returns A pointer to the supremum of this variable.

15.558.3.9 supremum()

```
template<typename T>
const Bound<T>& smtrat::vb::Variable< T >::supremum ( ) const [inline]
```

Returns A constant reference to the supremum of this variable.

15.558.3.10 updateBounds()

```
template<typename T>
bool smtrat::vb::Variable< T >::updateBounds (
            const Bound< T > & _changedBound )
```

Updates the infimum and supremum of this variable.

Parameters

_changedBound	The bound, for which we certainly know that it got deactivated before.

Returns true, if there is a conflict; false, otherwise.

15.558.3.11 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

15.558.3.12 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

15.558.3.13 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.

15.559 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 *&_changedVariable)

  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 smtrat::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

15.559.1 Detailed Description

template<typename T>
class smtrat::vb::VariableBounds< T >

Class to manage the bounds of a variable.

15.559.2 Member Typedef Documentation

15.559.2.1 ConstraintBoundMap template<typename T>

```
typedef std::map<ConstraintT,const Bound<T>*> smtrat::vb::VariableBounds< T >::ConstraintBoundMap
```

A map from Constraint pointers to Bound pointers.

15.559.2.2 VariableMap template<typename T>

```
typedef carl::FastMap<carl::Variable, Variable<T>*> smtrat::vb::VariableBounds< T >::VariableMap
```

A hash-map from arithmetic variables to variables managing the bounds.

15.559.3 Constructor & Destructor Documentation

15.559.3.1 VariableBounds() template<typename T>

```
smtrat::vb::VariableBounds< T >::VariableBounds ( )
```

Constructs a variable bounds manager.

15.559.3.2 ~VariableBounds() template<typename T>

```
smtrat::vb::VariableBounds< T >::~VariableBounds ( )
```

Destructs a variable bounds manager.

15.559.4 Member Function Documentation

15.559.4.1 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

_constraint	The constraint to consider.
_origin	The origin of the given constraint.

Returns true, if the variable bounds have been changed; false, otherwise.

15.559.4.2 addBound() [2/2] template<typename T>

```
bool smtrat::vb::VariableBounds< T >::addBound (
            const FormulaT & _formula,
            const T & _origin )
```

15.559.4.3 clear() template<typename T>

```
void smtrat::vb::VariableBounds< T >::clear ( )
```

15.559.4.4 empty() template<typename T>

```
bool smtrat::vb::VariableBounds< T >::empty ( ) const [inline]
```

15.559.4.5 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.

15.559.4.6 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.

15.559.4.7 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.

15.559.4.8 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.

15.559.4.9 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.

15.559.4.10 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.

15.559.4.11 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

_term	The term to compute the bounds as interval for.

Returns The term bounds as an interval.

15.559.4.12 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.

15.559.4.13 getOriginSetOfBounds() [1/3] template<typename T>

```
std::set<T> smtrat::vb::VariableBounds< T >::getOriginSetOfBounds ( ) const
```

15.559.4.14 getOriginSetOfBounds() [2/3] template<typename T>

```
std::set<T> smtrat::vb::VariableBounds< T >::getOriginSetOfBounds (
            const carl::Variable & _var ) const
```

15.559.4.15 getOriginSetOfBounds() [3/3] template<typename T>

```
std::set<T> smtrat::vb::VariableBounds< T >::getOriginSetOfBounds (
            const carl::Variables & _variables ) const
```

15.559.4.16 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.

15.559.4.17 getOriginsOfBounds() [2/3] template<typename T>

```
std::vector<T> smtrat::vb::VariableBounds< T >::getOriginsOfBounds (
            const carl::Variable & _var ) const
```

Parameters

_var	The variable to get origins of the bounds for.

Returns The origin constraints of the supremum and infimum of the given variable.

15.559.4.18 getOriginsOfBounds() [3/3] template<typename T>

```
std::vector<T> smtrat::vb::VariableBounds< T >::getOriginsOfBounds (
            const carl::Variables & _variables ) const
```

Parameters

_variables	The variables to get origins of the bounds for.

Returns The origin constraints of the supremum and infimum of the given variables.

15.559.4.19 isConflicting() template<typename T>

```
bool smtrat::vb::VariableBounds< T >::isConflicting ( ) const [inline]
```

Returns true, if there is a conflicting variable; false, otherwise.

15.559.4.20 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

_out	The output stream to print on.
_init	A string which is displayed at the beginning of every row.
_printAllBounds	A flag that indicates whether to print also the bounds of each variable (=true).

15.559.4.21 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

_constraint	The constraint, which effects shall be undone for the variable bounds.
_origin	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.

15.559.4.22 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

_constraint	The constraint, which effects shall be undone for the variable bounds.
_origin	The origin of the given constraint.
_changedVariable	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.

15.559.4.23 removeBound() [3/3]

```
template<typename T>
unsigned smtrat::vb::VariableBounds< T >::removeBound (
            const FormulaT & _formula,
            const T & _origin )
```

15.560 smtrat::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

15.560.1 Constructor & Destructor Documentation

15.560.1.1 VariableCapsule()

```
smtrat::VariableCapsule::VariableCapsule (
            const carl::Variable & xVariable,
            const carl::Variable & yVariable,
            const carl::Variable & zVariable )
```

15.560.2 Member Function Documentation

15.560.2.1 getXVariable()

```
const carl::Variable & smtrat::VariableCapsule::getXVariable ( )
const
```

15.560.2.2 getYVariable()

```
const carl::Variable & smtrat::VariableCapsule::getYVariable ( )
const
```

15.560.2.3 getZVariable()

```
const carl::Variable & smtrat::VariableCapsule::getZVariable ( ) const
```

15.560.3 Field Documentation

15.560.3.1 xVariable

```
carl::Variable smtrat::VariableCapsule::xVariable
```

15.560.3.2 yVariable

```
carl::Variable smtrat::VariableCapsule::yVariable
```

15.560.3.3 zVariable

```
carl::Variable smtrat::VariableCapsule::zVariable
```

15.561 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

15.561.1 Member Function Documentation

15.561.1.1 operator[]() [1/2]

```
std::size_t smtrat::mcsat::variableordering::VariableIDs::operator[]
(
    carl::Variable v ) [inline]
```

15.561.1.2 operator[]() [2/2]

```
std::size_t smtrat::mcsat::variableordering::VariableIDs::operator[]
(
    carl::Variable v ) const [inline]
```

15.561.2 Field Documentation

15.561.2.1 mIDs

```
std::map<carl::Variable, std::size_t> smtrat::mcsat::variableordering::↩
VariableIDs::mIDs
```

15.562 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.

15.562.1 Constructor & Destructor Documentation

15.562.1.1 VariableRewriteRule()

```
smtrat::VariableRewriteRule::VariableRewriteRule (
    unsigned varNr,
    const TermT & term,
    const carl::BitVector & reasons ) [inline]
```

15.562.1.2 ~VariableRewriteRule()

```
smtrat::VariableRewriteRule::~VariableRewriteRule ( ) [inline]
```

15.562.2 Field Documentation

15.562.2.1 mReasons

```
carl::BitVector smtrat::VariableRewriteRule::mReasons [protected]
```

Based on this origins.

15.562.2.2 mTerm `TermT smtrat::VariableRewriteRule::mTerm [protected]`

Rewrite with this term.

15.562.2.3 mVarNr `unsigned smtrat::VariableRewriteRule::mVarNr [protected]`

Rewrite a variable (identified by this number)

15.563 smtrat::mcsat::VariableSelector Class Reference

`#include <VariableSelector.h>`

Public Member Functions

- void add (const FormulaT &formula)
- void add (const carl::carlVariables &vars)
- void add (carl::Variable v)
- void remove (const carl::Variables &vars)
- void remove (carl::Variable v)
- void assign (carl::Variable v)
- void unassign (carl::Variable v)
- bool empty () const
- carl::Variable top () const

Friends

- std::ostream & operator<< (std::ostream &os, const VariableSelector &vs)

15.563.1 Member Function Documentation

15.563.1.1 add() [1/3] `void smtrat::mcsat::VariableSelector::add ( carl::Variable v ) [inline]`

15.563.1.2 add() [2/3] `void smtrat::mcsat::VariableSelector::add ( const carl::carlVariables & vars ) [inline]`

15.563.1.3 add() [3/3] `void smtrat::mcsat::VariableSelector::add ( const FormulaT & formula ) [inline]`

15.563.1.4 assign() `void smtrat::mcsat::VariableSelector::assign ( carl::Variable v ) [inline]`

15.563.1.5 empty() `bool smtrat::mcsat::VariableSelector::empty ( ) const [inline]`

15.563.1.6 remove() [1/2] `void smtrat::mcsat::VariableSelector::remove ( carl::Variable v ) [inline]`

15.563.1.7 remove() [2/2]

```
void smtrat::mcsat::VariableSelector::remove (
    const carl::Variables & vars ) [inline]
```

15.563.1.8 top()

```
carl::Variable smtrat::mcsat::VariableSelector::top ( ) const [inline]
```

15.563.1.9 unassign()

```
void smtrat::mcsat::VariableSelector::unassign (
    carl::Variable v ) [inline]
```

15.563.2 Friends And Related Function Documentation

15.563.2.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const VariableSelector & vs ) [friend]
```

15.564 smtrat::variant_hash_visitor Struct Reference

```
#include <VariantHash.h>
```

Public Member Functions

- template<typename T >
  std::size_t operator() (const T &value) const noexcept

15.564.1 Member Function Documentation

15.564.1.1 operator()()

```
template<typename T >
std::size_t smtrat::variant_hash_visitor::operator() (
    const T & value ) const [inline], [noexcept]
```

15.565 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

15.565.1 Detailed Description

template<typename Res>
struct smtrat::parser::conversion::VariantConverter< Res >

Converts variants to some type using the Converter class.

15.565.2 Member Typedef Documentation

15.565.2.1 result_type template<typename Res>

```
typedef bool smtrat::parser::conversion::VariantConverter< Res >::result_type
```

15.565.3 Member Function Documentation

15.565.3.1 convert() template<typename Res>

```
template<typename T >
Res smtrat::parser::conversion::VariantConverter< Res >::convert (
            const T & t )  [inline]
```

15.565.3.2 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]
```

15.565.3.3 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]
```

15.565.3.4 operator()() [3/3] template<typename Res>

```
template<typename T >
bool smtrat::parser::conversion::VariantConverter< Res >::operator() (
            const T & t )  [inline]
```

15.565.4 Field Documentation

15.565.4.1 converter template<typename Res>

```
Converter<Res> smtrat::parser::conversion::VariantConverter< Res >::converter
```

15.565.4.2 result template<typename Res>

```
Res smtrat::parser::conversion::VariantConverter< Res >::result
```

15.566 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.

15.566.1 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

Key	Type of keys.
Value	Type of values.

15.566.2 Member Function Documentation

15.566.2.1 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

T	Type that the value should have.
Output	Type of out.

Parameters

key	Key.
out	Functor returning an output stream.

15.566.2.2 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

T	Type of the value.

Parameters

key	Key.

Returns Value of key as type `T`.

15.566.2.3 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

T	Type that the value should have.

Parameters

key	Key.

Returns If there is a value of this type.

15.566.3 Field Documentation

15.566.3.1 elements `T std::map< K, T >::elements [inherited]`

STL member.

15.566.3.2 keys `K std::map< K, T >::keys [inherited]`

STL member.

15.567 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)

15.567.1 Detailed Description

template<typename Res>
struct smtrat::parser::conversion::VariantVariantConverter< Res >

Converts variants to another variant type not using the Converter class.

15.567.2 Member Typedef Documentation

15.567.2.1 result_type `template<typename Res>`

```
typedef Res smtrat::parser::conversion::VariantVariantConverter< Res >::result_type
```

15.567.3 Member Function Documentation

15.567.3.1 convert() `template<typename Res>`

```
template<typename Variant >
Res smtrat::parser::conversion::VariantVariantConverter< Res >::convert (
            const Variant & t )  [inline]
```

15.567.3.2 operator()() `template<typename Res>`

```
template<typename T >
Res smtrat::parser::conversion::VariantVariantConverter< Res >::operator() (
            const T & t )  [inline]
```

15.568 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

15.568.1 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.

15.568.2 Constructor & Destructor Documentation

15.568.2.1 VarSchedulerBase()

```
template<typename BaseModule >
smtrat::VarSchedulerBase::VarSchedulerBase (
            BaseModule & baseModule ) [inline]
```

15.568.3 Member Function Documentation

15.568.3.1 attachClause()

```
void smtrat::VarSchedulerBase::attachClause (
            Minisat::CRef ) [inline]
```

15.568.3.2 decreaseActivity()

```
void smtrat::VarSchedulerBase::decreaseActivity (
    Minisat::Var ) [inline]
```

15.568.3.3 detachClause()

```
void smtrat::VarSchedulerBase::detachClause (
    Minisat::CRef ) [inline]
```

15.568.3.4 empty()

```
bool smtrat::VarSchedulerBase::empty ( ) [inline]
```

Check if empty.

15.568.3.5 increaseActivity()

```
void smtrat::VarSchedulerBase::increaseActivity (
    Minisat::Var ) [inline]
```

15.568.3.6 insert()

```
void smtrat::VarSchedulerBase::insert (
    Minisat::Var ) [inline]
```

Insert a variable.

If already contained, do nothing.

15.568.3.7 pop()

```
Minisat::Lit smtrat::VarSchedulerBase::pop ( ) [inline]
```

Returns the next variable to be decided.

Returns and removes the next variable to be decided.

15.568.3.8 print()

```
void smtrat::VarSchedulerBase::print ( ) const [inline]
```

Check if variable is contained.

Print.

15.568.3.9 rebuild()

```
void smtrat::VarSchedulerBase::rebuild ( ) [inline]
```

Rebuild heap.

15.568.3.10 rebuildActivities()

```
void smtrat::VarSchedulerBase::rebuildActivities ( ) [inline]
```

15.568.3.11 rebuildTheoryVars()

```
template<typename Constraints >
void smtrat::VarSchedulerBase::rebuildTheoryVars (
    const Constraints & ) [inline]
```

15.568.3.12 relocateClauses()

```
void smtrat::VarSchedulerBase::relocateClauses (
    std::function< void(Minisat::CRef &)> ) [inline]
```

15.568.4 Field Documentation

15.568.4.1 abstractLiteral

`std::function<const Minisat::Lit(const FormulaT&)> smtrat::VarSchedulerBase::abstractLiteral [protected]`

15.568.4.2 abstractVariable

`std::function< Minisat::Var(const FormulaT&)> smtrat::VarSchedulerBase::abstractVariable [protected]`

15.568.4.3 currentlySatisfiedByBackend

`std::function<unsigned(const FormulaT&)> smtrat::VarSchedulerBase::currentlySatisfiedByBackend [protected]`

15.568.4.4 getActivity

`std::function<double(Minisat::Var)> smtrat::VarSchedulerBase::getActivity [protected]`

15.568.4.5 getBoolLitValue

`std::function<Minisat::lbool(Minisat::Lit)> smtrat::VarSchedulerBase::getBoolLitValue [protected]`

15.568.4.6 getBoolVarValue

`std::function<Minisat::lbool(Minisat::Var)> smtrat::VarSchedulerBase::getBoolVarValue [protected]`

15.568.4.7 getClause

`std::function<const Minisat::Clause&(Minisat::CRef)> smtrat::VarSchedulerBase::getClause [protected]`

15.568.4.8 getPolarity

`std::function<char(Minisat::Var)> smtrat::VarSchedulerBase::getPolarity [protected]`

15.568.4.9 isAbstractedFormula

`std::function<bool(const FormulaT&)> smtrat::VarSchedulerBase::isAbstractedFormula [protected]`

15.568.4.10 isBoolValueUndef

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isBoolValueUndef [protected]`

15.568.4.11 isDecisionVar

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isDecisionVar [protected]`

15.568.4.12 isTheoryAbstraction `std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isTheoryAbstraction [protected]`

15.568.4.13 reabstractLiteral `std::function<const FormulaT&(Minisat::Lit)> smtrat::VarScheduler Base::reabstractLiteral [protected]`

15.568.4.14 reabstractVariable `std::function<const FormulaT&(Minisat::Var)> smtrat::VarScheduler Base::reabstractVariable [protected]`

15.568.4.15 setPolarity `std::function<void(Minisat::Var,bool)> smtrat::VarSchedulerBase::set Polarity [protected]`

15.569 smtrat::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

15.569.1 Constructor & Destructor Documentation

15.569.1.1 VarSchedulerFixedRandom()

```
template<typename BaseModule >
smtrat::VarSchedulerFixedRandom::VarSchedulerFixedRandom (
            BaseModule & baseModule ) [inline], [explicit]
```

15.569.2 Member Function Documentation

15.569.2.1 attachClause()

```
void smtrat::VarSchedulerBase::attachClause (
            Minisat::CRef ) [inline], [inherited]
```

15.569.2.2 decreaseActivity()

```
void smtrat::VarSchedulerMinisat::decreaseActivity (
            Minisat::Var var ) [inline], [inherited]
```

15.569.2.3 detachClause()

```
void smtrat::VarSchedulerBase::detachClause (
            Minisat::CRef ) [inline], [inherited]
```

15.569.2.4 empty()

```
bool smtrat::VarSchedulerMinisat::empty ( ) [inline], [inherited]
```

15.569.2.5 increaseActivity()

```
void smtrat::VarSchedulerMinisat::increaseActivity (
            Minisat::Var var ) [inline], [inherited]
```

15.569.2.6 insert()

```
void smtrat::VarSchedulerMinisat::insert (
            Minisat::Var var ) [inline], [inherited]
```

15.569.2.7 pop()

```
Minisat::Lit smtrat::VarSchedulerMinisat::pop ( ) [inline], [inherited]
```

15.569.2.8 print()

```
void smtrat::VarSchedulerMinisat::print ( ) const [inline], [inherited]
```

15.569.2.9 rebuild()

```
void smtrat::VarSchedulerMinisat::rebuild ( ) [inline], [inherited]
```

15.569.2.10 rebuildActivities()

```
void smtrat::VarSchedulerMinisat::rebuildActivities ( ) [inline],
[inherited]
```

15.569.2.11 rebuildTheoryVars()

```
template<typename Constraints >
void smtrat::VarSchedulerBase::rebuildTheoryVars (
            const Constraints & ) [inline], [inherited]
```

15.569.2.12 relocateClauses()

`void smtrat::VarSchedulerBase::relocateClauses ( std::function< void(Minisat::CRef &)> ) [inline], [inherited]`

15.569.2.13 top()

`Minisat::Var smtrat::VarSchedulerMinisat::top ( ) [inline], [inherited]`

15.569.2.14 valid()

`bool smtrat::VarSchedulerMinisat::valid ( Minisat::Var var ) [inline], [protected], [inherited]`

15.569.3 Field Documentation

15.569.3.1 abstractLiteral

`std::function<const Minisat::Lit(const FormulaT&)> smtrat::VarSchedulerBase::abstractLiteral [protected], [inherited]`

15.569.3.2 abstractVariable

`std::function< Minisat::Var(const FormulaT&)> smtrat::VarSchedulerBase::abstractVariable [protected], [inherited]`

15.569.3.3 currentlySatisfiedByBackend

`std::function<unsigned(const FormulaT&)> smtrat::VarSchedulerBase::currentlySatisfiedByBackend [protected], [inherited]`

15.569.3.4 getActivity

`std::function<double(Minisat::Var)> smtrat::VarSchedulerBase::getActivity [protected], [inherited]`

15.569.3.5 getBoolLitValue

`std::function<Minisat::lbool(Minisat::Lit)> smtrat::VarSchedulerBase::getBoolLitValue [protected], [inherited]`

15.569.3.6 getBoolVarValue

`std::function<Minisat::lbool(Minisat::Var)> smtrat::VarSchedulerBase::getBoolVarValue [protected], [inherited]`

15.569.3.7 getClause

`std::function<const Minisat::Clause&(Minisat::CRef)> smtrat::VarSchedulerBase::getClause [protected], [inherited]`

15.569.3.8 getPolarity

`std::function<char(Minisat::Var)> smtrat::VarSchedulerBase::getPolarity [protected], [inherited]`

15.569.3.9 isAbstractedFormula

`std::function<bool(const FormulaT&)> smtrat::VarSchedulerBase::isAbstractedFormula [protected], [inherited]`

15.569.3.10 isBoolValueUndef

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isBoolValueUndef [protected], [inherited]`

15.569.3.11 isDecisionVar

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isDecisionVar [protected], [inherited]`

15.569.3.12 isTheoryAbstraction

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isTheoryAbstraction [protected], [inherited]`

15.569.3.13 reabstractLiteral

`std::function<const FormulaT&(Minisat::Lit)> smtrat::VarScheduler Base::reabstractLiteral [protected], [inherited]`

15.569.3.14 reabstractVariable

`std::function<const FormulaT&(Minisat::Var)> smtrat::VarScheduler Base::reabstractVariable [protected], [inherited]`

15.569.3.15 setPolarity

`std::function<void(Minisat::Var,bool)> smtrat::VarSchedulerBase::set Polarity [protected], [inherited]`

15.570 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

15.570.1 Detailed Description

template<mcsat::VariableOrdering vot>
class smtrat::VarSchedulerMcsatActivityPreferTheory< vot >

Activity-based scheduler preferring initially theory variables.

15.570.2 Constructor & Destructor Documentation

15.570.2.1 VarSchedulerMcsatActivityPreferTheory()

```
template<mcsat::VariableOrdering vot>
template<typename BaseModule >
smtrat::VarSchedulerMcsatActivityPreferTheory< vot >::VarSchedulerMcsatActivityPreferTheory (
            BaseModule & baseModule )  [inline], [explicit]
```

15.570.3 Member Function Documentation

15.570.3.1 attachClause()

```
void smtrat::VarSchedulerBase::attachClause (
            Minisat::CRef  )  [inline], [inherited]
```

15.570.3.2 decreaseActivity()

```
template<mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatActivityPreferTheory< vot >::decreaseActivity (
            Minisat::Var var )  [inline]
```

15.570.3.3 detachClause()

```
void smtrat::VarSchedulerBase::detachClause (
            Minisat::CRef  )  [inline], [inherited]
```

15.570.3.4 empty()

```
template<mcsat::VariableOrdering vot>
bool smtrat::VarSchedulerMcsatActivityPreferTheory< vot >::empty ( )  [inline]
```

15.570.3.5 increaseActivity()

```
template<mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatActivityPreferTheory< vot >::increaseActivity (
            Minisat::Var var ) [inline]
```

15.570.3.6 insert()

```
template<mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatActivityPreferTheory< vot >::insert (
            Minisat::Var var ) [inline]
```

15.570.3.7 pop()

```
template<mcsat::VariableOrdering vot>
Minisat::Lit smtrat::VarSchedulerMcsatActivityPreferTheory< vot >::pop ( ) [inline]
```

15.570.3.8 print()

```
template<mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatActivityPreferTheory< vot >::print ( ) const [inline]
```

15.570.3.9 rebuild()

```
template<mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatActivityPreferTheory< vot >::rebuild ( ) [inline]
```

15.570.3.10 rebuildActivities()

```
template<mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatActivityPreferTheory< vot >::rebuildActivities ( ) [inline]
```

15.570.3.11 rebuildTheoryVars()

```
template<mcsat::VariableOrdering vot>
template<typename Constraints >
void smtrat::VarSchedulerMcsatActivityPreferTheory< vot >::rebuildTheoryVars (
            const Constraints & c ) [inline]
```

15.570.3.12 relocateClauses()

```
void smtrat::VarSchedulerBase::relocateClauses (
            std::function< void(Minisat::CRef &)> ) [inline], [inherited]
```

15.570.4 Field Documentation

15.570.4.1 abstractLiteral

```
std::function<const Minisat::Lit(const FormulaT&)> smtrat::Var↩
SchedulerBase::abstractLiteral [protected], [inherited]
```

15.570.4.2 abstractVariable

```
std::function< Minisat::Var(const FormulaT&)> smtrat::VarScheduler↩
Base::abstractVariable [protected], [inherited]
```

15.570.4.3 carlVar

```
std::function<carl::Variable(Minisat::Var)> smtrat::VarSchedulerMcsatBase↩
::carlVar [protected], [inherited]
```

15.570.4.4 currentlySatisfiedByBackend

`std::function<unsigned(const FormulaT&)> smtrat::VarSchedulerBase::currentlySatisfiedByBackend [protected], [inherited]`

15.570.4.5 currentModel

`std::function<Model()> smtrat::VarSchedulerMcsatBase::currentModel [protected], [inherited]`

15.570.4.6 getActivity

`std::function<double(Minisat::Var)> smtrat::VarSchedulerBase::getActivity [protected], [inherited]`

15.570.4.7 getBoolLitValue

`std::function<Minisat::lbool(Minisat::Lit)> smtrat::VarSchedulerBase::getBoolLitValue [protected], [inherited]`

15.570.4.8 getBoolVarValue

`std::function<Minisat::lbool(Minisat::Var)> smtrat::VarSchedulerBase::getBoolVarValue [protected], [inherited]`

15.570.4.9 getClause

`std::function<const Minisat::Clause&(Minisat::CRef)> smtrat::VarSchedulerBase::getClause [protected], [inherited]`

15.570.4.10 getPolarity

`std::function<char(Minisat::Var)> smtrat::VarSchedulerBase::getPolarity [protected], [inherited]`

15.570.4.11 isAbstractedFormula

`std::function<bool(const FormulaT&)> smtrat::VarSchedulerBase::isAbstractedFormula [protected], [inherited]`

15.570.4.12 isBoolValueUndef

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isBoolValueUndef [protected], [inherited]`

15.570.4.13 isDecisionVar

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isDecisionVar [protected], [inherited]`

15.570.4.14 isTheoryAbstraction

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isTheoryAbstraction [protected], [inherited]`

15.570.4.15 isTheoryVar

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerMcsatBase::isTheoryVar [protected], [inherited]`

15.570.4.16 minisatVar `std::function<Minisat::Var(carl::Variable)> smtrat::VarSchedulerMcsat↩ Base::minisatVar [protected], [inherited]`

15.570.4.17 reabstractLiteral `std::function<const FormulaT&(Minisat::Lit)> smtrat::VarScheduler↩ Base::reabstractLiteral [protected], [inherited]`

15.570.4.18 reabstractVariable `std::function<const FormulaT&(Minisat::Var)> smtrat::VarScheduler↩ Base::reabstractVariable [protected], [inherited]`

15.570.4.19 setPolarity `std::function<void(Minisat::Var,bool)> smtrat::VarSchedulerBase::set↩ Polarity [protected], [inherited]`

15.571 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::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

15.571.1 Detailed Description

Base class for all MCSAT variable scheduler.

Should not be used directly.

15.571.2 Constructor & Destructor Documentation

15.571.2.1 VarSchedulerMcsatBase()

```
template<typename BaseModule >
smtrat::VarSchedulerMcsatBase::VarSchedulerMcsatBase (
            BaseModule & baseModule ) [inline]
```

15.571.3 Member Function Documentation

15.571.3.1 attachClause()

```
void smtrat::VarSchedulerBase::attachClause (
            Minisat::CRef ) [inline], [inherited]
```

15.571.3.2 decreaseActivity()

```
void smtrat::VarSchedulerBase::decreaseActivity (
            Minisat::Var ) [inline], [inherited]
```

15.571.3.3 detachClause()

```
void smtrat::VarSchedulerBase::detachClause (
            Minisat::CRef ) [inline], [inherited]
```

15.571.3.4 empty()

```
bool smtrat::VarSchedulerBase::empty ( ) [inline], [inherited]
```

Check if empty.

15.571.3.5 increaseActivity()

```
void smtrat::VarSchedulerBase::increaseActivity (
            Minisat::Var ) [inline], [inherited]
```

15.571.3.6 insert()

```
void smtrat::VarSchedulerBase::insert (
            Minisat::Var ) [inline], [inherited]
```

Insert a variable.

If already contained, do nothing.

15.571.3.7 pop()

```
Minisat::Lit smtrat::VarSchedulerBase::pop ( ) [inline], [inherited]
```

Returns the next variable to be decided.

Returns and removes the next variable to be decided.

15.571.3.8 print()

```
void smtrat::VarSchedulerBase::print ( ) const [inline], [inherited]
```

Check if variable is contained.

Print.

15.571.3.9 rebuild() `void smtrat::VarSchedulerBase::rebuild ( ) [inline], [inherited]`

Rebuild heap.

15.571.3.10 rebuildActivities() `void smtrat::VarSchedulerBase::rebuildActivities ( ) [inline], [inherited]`

15.571.3.11 rebuildTheoryVars() `template<typename Constraints >`
`void smtrat::VarSchedulerBase::rebuildTheoryVars (`
`const Constraints & ) [inline], [inherited]`

15.571.3.12 relocateClauses() `void smtrat::VarSchedulerBase::relocateClauses (`
`std::function< void(Minisat::CRef &)> ) [inline], [inherited]`

15.571.4 Field Documentation

15.571.4.1 abstractLiteral `std::function<const Minisat::Lit(const FormulaT&)> smtrat::VarSchedulerBase::abstractLiteral [protected], [inherited]`

15.571.4.2 abstractVariable `std::function< Minisat::Var(const FormulaT&)> smtrat::VarSchedulerBase::abstractVariable [protected], [inherited]`

15.571.4.3 carlVar `std::function<carl::Variable(Minisat::Var)> smtrat::VarSchedulerMcsatBase::carlVar [protected]`

15.571.4.4 currentlySatisfiedByBackend `std::function<unsigned(const FormulaT&)> smtrat::VarSchedulerBase::currentlySatisfiedByBackend [protected], [inherited]`

15.571.4.5 currentModel `std::function<Model()> smtrat::VarSchedulerMcsatBase::currentModel [protected]`

15.571.4.6 getActivity `std::function<double(Minisat::Var)> smtrat::VarSchedulerBase::getActivity [protected], [inherited]`

15.571.4.7 getBoolLitValue `std::function<Minisat::lbool(Minisat::Lit)> smtrat::VarSchedulerBase::getBoolLitValue [protected], [inherited]`

15.571.4.8 getBoolVarValue `std::function<Minisat::lbool(Minisat::Var)> smtrat::VarScheduler↩ Base::getBoolVarValue [protected], [inherited]`

15.571.4.9 getClause `std::function<const Minisat::Clause&(Minisat::CRef)> smtrat::VarScheduler↩ Base::getClause [protected], [inherited]`

15.571.4.10 getPolarity `std::function<char(Minisat::Var)> smtrat::VarSchedulerBase::getPolarity [protected], [inherited]`

15.571.4.11 isAbstractedFormula `std::function<bool(const FormulaT&)> smtrat::VarScheduler↩ Base::isAbstractedFormula [protected], [inherited]`

15.571.4.12 isBoolValueUndef `std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::is↩ BoolValueUndef [protected], [inherited]`

15.571.4.13 isDecisionVar `std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::is↩ DecisionVar [protected], [inherited]`

15.571.4.14 isTheoryAbstraction `std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase↩ ::isTheoryAbstraction [protected], [inherited]`

15.571.4.15 isTheoryVar `std::function<bool(Minisat::Var)> smtrat::VarSchedulerMcsatBase::is↩ TheoryVar [protected]`

15.571.4.16 minisatVar `std::function<Minisat::Var(carl::Variable)> smtrat::VarSchedulerMcsat↩ Base::minisatVar [protected]`

15.571.4.17 reabstractLiteral `std::function<const FormulaT&(Minisat::Lit)> smtrat::VarScheduler↩ Base::reabstractLiteral [protected], [inherited]`

15.571.4.18 reabstractVariable `std::function<const FormulaT&(Minisat::Var)> smtrat::VarScheduler↩ Base::reabstractVariable [protected], [inherited]`

15.571.4.19 setPolarity `std::function<void(Minisat::Var,bool)> smtrat::VarSchedulerBase::set↩ Polarity [protected], [inherited]`

15.572 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

15.572.1 Detailed Description

template<mcsat::VariableOrdering vot>
class smtrat::VarSchedulerMcsatBooleanFirst< vot >

Variable scheduling that all decides Boolean variables first before deciding any theory variable.

15.572.2 Constructor & Destructor Documentation

15.572.2.1 VarSchedulerMcsatBooleanFirst()

```
template<mcsat::VariableOrdering vot>
template<typename BaseModule >
smtrat::VarSchedulerMcsatBooleanFirst< vot >::VarSchedulerMcsatBooleanFirst (
            BaseModule & baseModule ) [inline], [explicit]
```

15.572.3 Member Function Documentation

15.572.3.1 attachClause()

```
void smtrat::VarSchedulerBase::attachClause (
            Minisat::CRef  )  [inline], [inherited]
```

15.572.3.2 decreaseActivity()

```
template<mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatBooleanFirst< vot >::decreaseActivity (
            Minisat::Var var )  [inline]
```

15.572.3.3 detachClause()

```
void smtrat::VarSchedulerBase::detachClause (
            Minisat::CRef  )  [inline], [inherited]
```

15.572.3.4 empty()

```
template<mcsat::VariableOrdering vot>
bool smtrat::VarSchedulerMcsatBooleanFirst< vot >::empty ( )  [inline]
```

15.572.3.5 increaseActivity()

```
template<mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatBooleanFirst< vot >::increaseActivity (
            Minisat::Var var )  [inline]
```

15.572.3.6 insert()

```
template<mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatBooleanFirst< vot >::insert (
            Minisat::Var var )  [inline]
```

15.572.3.7 pop()

```
template<mcsat::VariableOrdering vot>
Minisat::Lit smtrat::VarSchedulerMcsatBooleanFirst< vot >::pop ( )  [inline]
```

15.572.3.8 print()

```
template<mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatBooleanFirst< vot >::print ( ) const  [inline]
```

15.572.3.9 rebuild()

```
template<mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatBooleanFirst< vot >::rebuild ( )  [inline]
```

15.572.3.10 rebuildActivities()

```
template<mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatBooleanFirst< vot >::rebuildActivities ( )  [inline]
```

15.572.3.11 rebuildTheoryVars()

```
template<mcsat::VariableOrdering vot>
template<typename Constraints >
void smtrat::VarSchedulerMcsatBooleanFirst< vot >::rebuildTheoryVars (
            const Constraints & c )  [inline]
```

15.572.3.12 relocateClauses()

```
void smtrat::VarSchedulerBase::relocateClauses (
    std::function< void(Minisat::CRef &)> ) [inline], [inherited]
```

15.572.4 Field Documentation

15.572.4.1 abstractLiteral

```
std::function<const Minisat::Lit(const FormulaT&)> smtrat::VarSchedulerBase::abstractLiteral [protected], [inherited]
```

15.572.4.2 abstractVariable

```
std::function< Minisat::Var(const FormulaT&)> smtrat::VarSchedulerBase::abstractVariable [protected], [inherited]
```

15.572.4.3 carlVar

```
std::function<carl::Variable(Minisat::Var)> smtrat::VarSchedulerMcsatBase::carlVar [protected], [inherited]
```

15.572.4.4 currentlySatisfiedByBackend

```
std::function<unsigned(const FormulaT&)> smtrat::VarSchedulerBase::currentlySatisfiedByBackend [protected], [inherited]
```

15.572.4.5 currentModel

```
std::function<Model()> smtrat::VarSchedulerMcsatBase::currentModel [protected], [inherited]
```

15.572.4.6 getActivity

```
std::function<double(Minisat::Var)> smtrat::VarSchedulerBase::getActivity [protected], [inherited]
```

15.572.4.7 getBoolLitValue

```
std::function<Minisat::lbool(Minisat::Lit)> smtrat::VarSchedulerBase::getBoolLitValue [protected], [inherited]
```

15.572.4.8 getBoolVarValue

```
std::function<Minisat::lbool(Minisat::Var)> smtrat::VarSchedulerBase::getBoolVarValue [protected], [inherited]
```

15.572.4.9 getClause

```
std::function<const Minisat::Clause&(Minisat::CRef)> smtrat::VarSchedulerBase::getClause [protected], [inherited]
```

15.572.4.10 getPolarity

```
std::function<char(Minisat::Var)> smtrat::VarSchedulerBase::getPolarity [protected], [inherited]
```

15.572.4.11 isAbstractedFormula

```
std::function<bool(const FormulaT&)> smtrat::VarSchedulerBase::isAbstractedFormula [protected], [inherited]
```

15.572.4.12 isBoolValueUndef `std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isBoolValueUndef [protected], [inherited]`

15.572.4.13 isDecisionVar `std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isDecisionVar [protected], [inherited]`

15.572.4.14 isTheoryAbstraction `std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isTheoryAbstraction [protected], [inherited]`

15.572.4.15 isTheoryVar `std::function<bool(Minisat::Var)> smtrat::VarSchedulerMcsatBase::isTheoryVar [protected], [inherited]`

15.572.4.16 minisatVar `std::function<Minisat::Var(carl::Variable)> smtrat::VarSchedulerMcsatBase::minisatVar [protected], [inherited]`

15.572.4.17 reabstractLiteral `std::function<const FormulaT&(Minisat::Lit)> smtrat::VarSchedulerBase::reabstractLiteral [protected], [inherited]`

15.572.4.18 reabstractVariable `std::function<const FormulaT&(Minisat::Var)> smtrat::VarSchedulerBase::reabstractVariable [protected], [inherited]`

15.572.4.19 setPolarity `std::function<void(Minisat::Var,bool)> smtrat::VarSchedulerBase::setPolarity [protected], [inherited]`

15.573 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

15.573.1 Detailed Description

template<typename TheoryScheduler>
class smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >

Variable scheduling that all decides theory variables first before deciding any Boolean variable.

15.573.2 Constructor & Destructor Documentation

15.573.2.1 VarSchedulerMcsatTheoryFirst()

```
template<typename TheoryScheduler >
template<typename BaseModule >
smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::VarSchedulerMcsatTheoryFirst (
            BaseModule & baseModule ) [inline], [explicit]
```

15.573.3 Member Function Documentation

15.573.3.1 attachClause()

```
void smtrat::VarSchedulerBase::attachClause (
            Minisat::CRef ) [inline], [inherited]
```

15.573.3.2 decreaseActivity()

```
template<typename TheoryScheduler >
void smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::decreaseActivity (
            Minisat::Var var ) [inline]
```

15.573.3.3 detachClause()

```
void smtrat::VarSchedulerBase::detachClause (
            Minisat::CRef ) [inline], [inherited]
```

15.573.3.4 empty()

```
template<typename TheoryScheduler >
bool smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::empty ( ) [inline]
```

15.573.3.5 increaseActivity()

```
template<typename TheoryScheduler >
void smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::increaseActivity (
            Minisat::Var var )  [inline]
```

15.573.3.6 insert()

```
template<typename TheoryScheduler >
void smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::insert (
            Minisat::Var var )  [inline]
```

15.573.3.7 pop()

```
template<typename TheoryScheduler >
Minisat::Lit smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::pop ( )  [inline]
```

15.573.3.8 print()

```
template<typename TheoryScheduler >
void smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::print ( ) const  [inline]
```

15.573.3.9 rebuild()

```
template<typename TheoryScheduler >
void smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::rebuild ( )  [inline]
```

15.573.3.10 rebuildActivities()

```
template<typename TheoryScheduler >
void smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::rebuildActivities ( )  [inline]
```

15.573.3.11 rebuildTheoryVars()

```
template<typename TheoryScheduler >
template<typename Constraints >
void smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::rebuildTheoryVars (
            const Constraints & c )  [inline]
```

15.573.3.12 relocateClauses()

```
void smtrat::VarSchedulerBase::relocateClauses (
            std::function< void(Minisat::CRef &)> )  [inline], [inherited]
```

15.573.4 Field Documentation

15.573.4.1 abstractLiteral

```
std::function<const Minisat::Lit(const FormulaT&)> smtrat::Var↩
SchedulerBase::abstractLiteral  [protected], [inherited]
```

15.573.4.2 abstractVariable

```
std::function< Minisat::Var(const FormulaT&)> smtrat::VarScheduler↩
Base::abstractVariable  [protected], [inherited]
```

15.573.4.3 carlVar

```
std::function<carl::Variable(Minisat::Var)> smtrat::VarSchedulerMcsatBase↩
::carlVar  [protected], [inherited]
```

15.573.4.4 currentlySatisfiedByBackend

`std::function<unsigned(const FormulaT&)> smtrat::VarSchedulerBase::currentlySatisfiedByBackend [protected], [inherited]`

15.573.4.5 currentModel

`std::function<Model()> smtrat::VarSchedulerMcsatBase::currentModel [protected], [inherited]`

15.573.4.6 getActivity

`std::function<double(Minisat::Var)> smtrat::VarSchedulerBase::getActivity [protected], [inherited]`

15.573.4.7 getBoolLitValue

`std::function<Minisat::lbool(Minisat::Lit)> smtrat::VarSchedulerBase::getBoolLitValue [protected], [inherited]`

15.573.4.8 getBoolVarValue

`std::function<Minisat::lbool(Minisat::Var)> smtrat::VarSchedulerBase::getBoolVarValue [protected], [inherited]`

15.573.4.9 getClause

`std::function<const Minisat::Clause&(Minisat::CRef)> smtrat::VarSchedulerBase::getClause [protected], [inherited]`

15.573.4.10 getPolarity

`std::function<char(Minisat::Var)> smtrat::VarSchedulerBase::getPolarity [protected], [inherited]`

15.573.4.11 isAbstractedFormula

`std::function<bool(const FormulaT&)> smtrat::VarSchedulerBase::isAbstractedFormula [protected], [inherited]`

15.573.4.12 isBoolValueUndef

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isBoolValueUndef [protected], [inherited]`

15.573.4.13 isDecisionVar

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isDecisionVar [protected], [inherited]`

15.573.4.14 isTheoryAbstraction

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isTheoryAbstraction [protected], [inherited]`

15.573.4.15 isTheoryVar

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerMcsatBase::isTheoryVar [protected], [inherited]`

15.573.4.16 minisatVar `std::function<Minisat::Var(carl::Variable)> smtrat::VarSchedulerMcsatBase::minisatVar [protected], [inherited]`

15.573.4.17 reabstractLiteral `std::function<const FormulaT&(Minisat::Lit)> smtrat::VarSchedulerBase::reabstractLiteral [protected], [inherited]`

15.573.4.18 reabstractVariable `std::function<const FormulaT&(Minisat::Var)> smtrat::VarSchedulerBase::reabstractVariable [protected], [inherited]`

15.573.4.19 setPolarity `std::function<void(Minisat::Var,bool)> smtrat::VarSchedulerBase::setPolarity [protected], [inherited]`

15.574 smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities > Class Template Reference

Decides only Constraints occuring 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

15.574.1 Detailed Description

template<typename TheoryScheduler, bool respectActivities>
class smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >

Decides only Constraints occuring in clauses that are univariate in the current theory variable while the theory ordering is static.

This corresponds to the original NLSAT strategy.

15.574.2 Constructor & Destructor Documentation

15.574.2.1 VarSchedulerMcsatUnivariateClausesOnly()

```
template<typename TheoryScheduler , bool
respectActivities>
template<typename BaseModule >
smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >::VarSchedulerMcsatUnivar
(
            BaseModule & baseModule )  [inline], [explicit]
```

15.574.3 Member Function Documentation

15.574.3.1 attachClause()

```
template<typename TheoryScheduler , bool respectActivities>
void smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >↩
::attachClause (
            Minisat::CRef cl )  [inline]
```

15.574.3.2 decreaseActivity()

```
template<typename TheoryScheduler , bool respectActivities>
void smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >↩
::decreaseActivity (
            Minisat::Var var )  [inline]
```

15.574.3.3 detachClause()

```
template<typename TheoryScheduler , bool respectActivities>
void smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >↩
::detachClause (
            Minisat::CRef cl )  [inline]
```

15.574.3.4 empty()

```
template<typename TheoryScheduler , bool respectActivities>
bool smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >↩
::empty ( )  [inline]
```

15.574.3.5 increaseActivity()

```
template<typename TheoryScheduler , bool respectActivities>
void smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >↩
::increaseActivity (
            Minisat::Var var )  [inline]
```

15.574.3.6 insert()

```
template<typename TheoryScheduler , bool respectActivities>
void smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >↩
::insert (
            Minisat::Var var )  [inline]
```

15.574.3.7 pop()

```
template<typename TheoryScheduler , bool respectActivities>
Minisat::Lit smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities
>::pop ( )  [inline]
```

15.574.3.8 print()

```
template<typename TheoryScheduler , bool respectActivities>
void smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >↩
::print ( ) const  [inline]
```

15.574.3.9 rebuild()

```
template<typename TheoryScheduler , bool respectActivities>
void smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >↩
::rebuild ( )  [inline]
```

15.574.3.10 rebuildActivities()

```
template<typename TheoryScheduler , bool respectActivities>
void smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >↩
::rebuildActivities ( )  [inline]
```

15.574.3.11 rebuildTheoryVars()

```
template<typename TheoryScheduler , bool respectActivities>
template<typename Constraints >
void smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >↩
::rebuildTheoryVars (
            const Constraints & c )  [inline]
```

15.574.3.12 relocateClauses()

```
template<typename TheoryScheduler , bool respectActivities>
void smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >↩
::relocateClauses (
            std::function< void(Minisat::CRef &)> relocate )  [inline]
```

15.574.4 Field Documentation

15.574.4.1 abstractLiteral

```
std::function<const Minisat::Lit(const FormulaT&)> smtrat::Var↩
SchedulerBase::abstractLiteral  [protected], [inherited]
```

15.574.4.2 abstractVariable

```
std::function< Minisat::Var(const FormulaT&)> smtrat::VarScheduler↩
Base::abstractVariable  [protected], [inherited]
```

15.574.4.3 carlVar

```
std::function<carl::Variable(Minisat::Var)> smtrat::VarSchedulerMcsatBase↩
::carlVar  [protected], [inherited]
```

15.574.4.4 currentlySatisfiedByBackend

`std::function<unsigned(const FormulaT&)> smtrat::VarSchedulerBase::currentlySatisfiedByBackend [protected], [inherited]`

15.574.4.5 currentModel

`std::function<Model()> smtrat::VarSchedulerMcsatBase::currentModel [protected], [inherited]`

15.574.4.6 getActivity

`std::function<double(Minisat::Var)> smtrat::VarSchedulerBase::getActivity [protected], [inherited]`

15.574.4.7 getBoolLitValue

`std::function<Minisat::lbool(Minisat::Lit)> smtrat::VarSchedulerBase::getBoolLitValue [protected], [inherited]`

15.574.4.8 getBoolVarValue

`std::function<Minisat::lbool(Minisat::Var)> smtrat::VarSchedulerBase::getBoolVarValue [protected], [inherited]`

15.574.4.9 getClause

`std::function<const Minisat::Clause&(Minisat::CRef)> smtrat::VarSchedulerBase::getClause [protected], [inherited]`

15.574.4.10 getPolarity

`std::function<char(Minisat::Var)> smtrat::VarSchedulerBase::getPolarity [protected], [inherited]`

15.574.4.11 isAbstractedFormula

`std::function<bool(const FormulaT&)> smtrat::VarSchedulerBase::isAbstractedFormula [protected], [inherited]`

15.574.4.12 isBoolValueUndef

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isBoolValueUndef [protected], [inherited]`

15.574.4.13 isDecisionVar

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isDecisionVar [protected], [inherited]`

15.574.4.14 isTheoryAbstraction

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isTheoryAbstraction [protected], [inherited]`

15.574.4.15 isTheoryVar

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerMcsatBase::isTheoryVar [protected], [inherited]`

15.574.4.16 minisatVar `std::function<Minisat::Var(carl::Variable)> smtrat::VarSchedulerMcsatBase::minisatVar [protected], [inherited]`

15.574.4.17 reabstractLiteral `std::function<const FormulaT&(Minisat::Lit)> smtrat::VarSchedulerBase::reabstractLiteral [protected], [inherited]`

15.574.4.18 reabstractVariable `std::function<const FormulaT&(Minisat::Var)> smtrat::VarSchedulerBase::reabstractVariable [protected], [inherited]`

15.574.4.19 setPolarity `std::function<void(Minisat::Var,bool)> smtrat::VarSchedulerBase::setPolarity [protected], [inherited]`

15.575 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

15.575.1 Detailed Description

template<int lookahead, mcsat::VariableOrdering vot>
class smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >

Decides only Constraints univariate in the current theory variable while the theory ordering is static.

15.575.2 Constructor & Destructor Documentation

15.575.2.1 VarSchedulerMcsatUnivariateConstraintsOnly()

```
template<int lookahead, mcsat::Variable↩
Ordering vot>
template<typename BaseModule >
smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::VarSchedulerMcsatUnivariateConstraintsO
(
            BaseModule & baseModule )  [inline], [explicit]
```

15.575.3 Member Function Documentation

15.575.3.1 attachClause()

```
void smtrat::VarSchedulerBase::attachClause (
            Minisat::CRef  )  [inline], [inherited]
```

15.575.3.2 decreaseActivity()

```
template<int lookahead, mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::decreaseActivity (
            Minisat::Var var )  [inline]
```

15.575.3.3 detachClause()

```
void smtrat::VarSchedulerBase::detachClause (
            Minisat::CRef  )  [inline], [inherited]
```

15.575.3.4 empty()

```
template<int lookahead, mcsat::VariableOrdering vot>
bool smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::empty ( )  [inline]
```

15.575.3.5 increaseActivity()

```
template<int lookahead, mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::increaseActivity (
            Minisat::Var var )  [inline]
```

15.575.3.6 insert()

```
template<int lookahead, mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::insert (
            Minisat::Var var )  [inline]
```

15.575.3.7 pop()

```
template<int lookahead, mcsat::VariableOrdering vot>
Minisat::Lit smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::pop ( )
[inline]
```

15.575.3.8 print() template<int lookahead, mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::print ( ) const [inline]

15.575.3.9 rebuild() template<int lookahead, mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::rebuild ( ) [inline]

15.575.3.10 rebuildActivities() template<int lookahead, mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::rebuildActivities ( ) [inline]

15.575.3.11 rebuildTheoryVars() template<int lookahead, mcsat::VariableOrdering vot>
template<typename Constraints >
void smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::rebuildTheoryVars (const Constraints & *c*) [inline]

15.575.3.12 relocateClauses() void smtrat::VarSchedulerBase::relocateClauses (std::function< void(Minisat::CRef &)>) [inline], [inherited]

15.575.4 Field Documentation

15.575.4.1 abstractLiteral std::function<const Minisat::Lit(const FormulaT&)> smtrat::Var↩SchedulerBase::abstractLiteral [protected], [inherited]

15.575.4.2 abstractVariable std::function< Minisat::Var(const FormulaT&)> smtrat::VarScheduler↩Base::abstractVariable [protected], [inherited]

15.575.4.3 carlVar std::function<carl::Variable(Minisat::Var)> smtrat::VarSchedulerMcsatBase↩::carlVar [protected], [inherited]

15.575.4.4 currentlySatisfiedByBackend std::function<unsigned(const FormulaT&)> smtrat::Var↩SchedulerBase::currentlySatisfiedByBackend [protected], [inherited]

15.575.4.5 currentModel std::function<Model()> smtrat::VarSchedulerMcsatBase::currentModel [protected], [inherited]

15.575.4.6 getActivity std::function<double(Minisat::Var)> smtrat::VarSchedulerBase::get↩Activity [protected], [inherited]

15.575.4.7 getBoolLitValue

`std::function<Minisat::lbool(Minisat::Lit)> smtrat::VarScheduler↩ Base::getBoolLitValue [protected], [inherited]`

15.575.4.8 getBoolVarValue

`std::function<Minisat::lbool(Minisat::Var)> smtrat::VarScheduler↩ Base::getBoolVarValue [protected], [inherited]`

15.575.4.9 getClause

`std::function<const Minisat::Clause&(Minisat::CRef)> smtrat::VarScheduler↩ Base::getClause [protected], [inherited]`

15.575.4.10 getPolarity

`std::function<char(Minisat::Var)> smtrat::VarSchedulerBase::getPolarity [protected], [inherited]`

15.575.4.11 isAbstractedFormula

`std::function<bool(const FormulaT&)> smtrat::VarScheduler↩ Base::isAbstractedFormula [protected], [inherited]`

15.575.4.12 isBoolValueUndef

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::is↩ BoolValueUndef [protected], [inherited]`

15.575.4.13 isDecisionVar

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::is↩ DecisionVar [protected], [inherited]`

15.575.4.14 isTheoryAbstraction

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase↩ ::isTheoryAbstraction [protected], [inherited]`

15.575.4.15 isTheoryVar

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerMcsatBase::is↩ TheoryVar [protected], [inherited]`

15.575.4.16 minisatVar

`std::function<Minisat::Var(carl::Variable)> smtrat::VarSchedulerMcsat↩ Base::minisatVar [protected], [inherited]`

15.575.4.17 reabstractLiteral

`std::function<const FormulaT&(Minisat::Lit)> smtrat::VarScheduler↩ Base::reabstractLiteral [protected], [inherited]`

15.575.4.18 reabstractVariable

`std::function<const FormulaT&(Minisat::Var)> smtrat::VarScheduler↩ Base::reabstractVariable [protected], [inherited]`

15.575.4.19 setPolarity `std::function<void(Minisat::Var,bool)> smtrat::VarSchedulerBase::setPolarity [protected], [inherited]`

15.576 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

15.576.1 Detailed Description

Minisat's activity-based variable scheduling.

15.576.2 Constructor & Destructor Documentation

15.576.2.1 VarSchedulerMinisat() [1/2]

```
template<typename BaseModule >
smtrat::VarSchedulerMinisat::VarSchedulerMinisat (
            BaseModule & baseModule,
            std::function< bool(Minisat::Var, Minisat::Var)> cmp ) [inline], [explicit]
```

15.576.2.2 VarSchedulerMinisat() [2/2]

```
template<typename BaseModule >
smtrat::VarSchedulerMinisat::VarSchedulerMinisat (
            BaseModule & baseModule ) [inline], [explicit]
```

15.576.3 Member Function Documentation

15.576.3.1 attachClause()

```
void smtrat::VarSchedulerBase::attachClause (
            Minisat::CRef ) [inline], [inherited]
```

15.576.3.2 decreaseActivity()

```
void smtrat::VarSchedulerMinisat::decreaseActivity (
            Minisat::Var var ) [inline]
```

15.576.3.3 detachClause()

```
void smtrat::VarSchedulerBase::detachClause (
            Minisat::CRef ) [inline], [inherited]
```

15.576.3.4 empty()

```
bool smtrat::VarSchedulerMinisat::empty ( ) [inline]
```

15.576.3.5 increaseActivity()

```
void smtrat::VarSchedulerMinisat::increaseActivity (
            Minisat::Var var ) [inline]
```

15.576.3.6 insert()

```
void smtrat::VarSchedulerMinisat::insert (
            Minisat::Var var ) [inline]
```

15.576.3.7 pop()

```
Minisat::Lit smtrat::VarSchedulerMinisat::pop ( ) [inline]
```

15.576.3.8 print()

```
void smtrat::VarSchedulerMinisat::print ( ) const [inline]
```

15.576.3.9 rebuild()

```
void smtrat::VarSchedulerMinisat::rebuild ( ) [inline]
```

15.576.3.10 rebuildActivities()

```
void smtrat::VarSchedulerMinisat::rebuildActivities ( ) [inline]
```

15.576.3.11 rebuildTheoryVars()

```
template<typename Constraints >
void smtrat::VarSchedulerBase::rebuildTheoryVars (
            const Constraints & ) [inline], [inherited]
```

15.576.3.12 relocateClauses()

`void smtrat::VarSchedulerBase::relocateClauses ( std::function< void(Minisat::CRef &)> ) [inline], [inherited]`

15.576.3.13 top()

`Minisat::Var smtrat::VarSchedulerMinisat::top ( ) [inline]`

15.576.3.14 valid()

`bool smtrat::VarSchedulerMinisat::valid ( Minisat::Var var ) [inline], [protected]`

15.576.4 Field Documentation

15.576.4.1 abstractLiteral

`std::function<const Minisat::Lit(const FormulaT&)> smtrat::VarSchedulerBase::abstractLiteral [protected], [inherited]`

15.576.4.2 abstractVariable

`std::function< Minisat::Var(const FormulaT&)> smtrat::VarSchedulerBase::abstractVariable [protected], [inherited]`

15.576.4.3 currentlySatisfiedByBackend

`std::function<unsigned(const FormulaT&)> smtrat::VarSchedulerBase::currentlySatisfiedByBackend [protected], [inherited]`

15.576.4.4 getActivity

`std::function<double(Minisat::Var)> smtrat::VarSchedulerBase::getActivity [protected], [inherited]`

15.576.4.5 getBoolLitValue

`std::function<Minisat::lbool(Minisat::Lit)> smtrat::VarSchedulerBase::getBoolLitValue [protected], [inherited]`

15.576.4.6 getBoolVarValue

`std::function<Minisat::lbool(Minisat::Var)> smtrat::VarSchedulerBase::getBoolVarValue [protected], [inherited]`

15.576.4.7 getClause

`std::function<const Minisat::Clause&(Minisat::CRef)> smtrat::VarSchedulerBase::getClause [protected], [inherited]`

15.576.4.8 getPolarity

`std::function<char(Minisat::Var)> smtrat::VarSchedulerBase::getPolarity [protected], [inherited]`

15.576.4.9 isAbstractedFormula

`std::function<bool(const FormulaT&)> smtrat::VarSchedulerBase::isAbstractedFormula [protected], [inherited]`

15.576.4.10 isBoolValueUndef

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isBoolValueUndef [protected], [inherited]`

15.576.4.11 isDecisionVar

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isDecisionVar [protected], [inherited]`

15.576.4.12 isTheoryAbstraction

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isTheoryAbstraction [protected], [inherited]`

15.576.4.13 reabstractLiteral

`std::function<const FormulaT&(Minisat::Lit)> smtrat::VarSchedulerBase::reabstractLiteral [protected], [inherited]`

15.576.4.14 reabstractVariable

`std::function<const FormulaT&(Minisat::Var)> smtrat::VarSchedulerBase::reabstractVariable [protected], [inherited]`

15.576.4.15 setPolarity

`std::function<void(Minisat::Var,bool)> smtrat::VarSchedulerBase::setPolarity [protected], [inherited]`

15.577 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

15.577.1 Detailed Description

Random scheduler.

15.577.2 Constructor & Destructor Documentation

15.577.2.1 VarSchedulerRandom()

```
template<typename BaseModule >
smtrat::VarSchedulerRandom::VarSchedulerRandom (
            BaseModule & baseModule ) [inline], [explicit]
```

15.577.3 Member Function Documentation

15.577.3.1 attachClause()

```
void smtrat::VarSchedulerBase::attachClause (
            Minisat::CRef ) [inline], [inherited]
```

15.577.3.2 decreaseActivity()

```
void smtrat::VarSchedulerBase::decreaseActivity (
            Minisat::Var ) [inline], [inherited]
```

15.577.3.3 detachClause()

```
void smtrat::VarSchedulerBase::detachClause (
            Minisat::CRef ) [inline], [inherited]
```

15.577.3.4 empty()

```
bool smtrat::VarSchedulerRandom::empty ( ) [inline]
```

15.577.3.5 increaseActivity()

```
void smtrat::VarSchedulerBase::increaseActivity (
            Minisat::Var ) [inline], [inherited]
```

15.577.3.6 insert()

```
void smtrat::VarSchedulerRandom::insert (
            Minisat::Var var ) [inline]
```

15.577.3.7 pop()

```
Minisat::Lit smtrat::VarSchedulerRandom::pop ( ) [inline]
```

15.577.3.8 print() `void smtrat::VarSchedulerRandom::print ( ) const [inline]`

15.577.3.9 rebuild() `void smtrat::VarSchedulerRandom::rebuild ( ) [inline]`

15.577.3.10 rebuildActivities() `void smtrat::VarSchedulerBase::rebuildActivities ( ) [inline], [inherited]`

15.577.3.11 rebuildTheoryVars()

```
template<typename Constraints >
void smtrat::VarSchedulerBase::rebuildTheoryVars (
            const Constraints & ) [inline], [inherited]
```

15.577.3.12 relocateClauses()

```
void smtrat::VarSchedulerBase::relocateClauses (
            std::function< void(Minisat::CRef &)> ) [inline], [inherited]
```

15.577.3.13 valid()

```
bool smtrat::VarSchedulerRandom::valid (
            Minisat::Var var ) [inline], [protected]
```

15.577.4 Field Documentation

15.577.4.1 abstractLiteral `std::function<const Minisat::Lit(const FormulaT&)> smtrat::VarSchedulerBase::abstractLiteral [protected], [inherited]`

15.577.4.2 abstractVariable `std::function< Minisat::Var(const FormulaT&)> smtrat::VarSchedulerBase::abstractVariable [protected], [inherited]`

15.577.4.3 currentlySatisfiedByBackend `std::function<unsigned(const FormulaT&)> smtrat::VarSchedulerBase::currentlySatisfiedByBackend [protected], [inherited]`

15.577.4.4 getActivity `std::function<double(Minisat::Var)> smtrat::VarSchedulerBase::getActivity [protected], [inherited]`

15.577.4.5 getBoolLitValue `std::function<Minisat::lbool(Minisat::Lit)> smtrat::VarSchedulerBase::getBoolLitValue [protected], [inherited]`

15.577.4.6 getBoolVarValue `std::function<Minisat::lbool(Minisat::Var)> smtrat::VarSchedulerBase::getBoolVarValue [protected], [inherited]`

15.577.4.7 getClause

`std::function<const Minisat::Clause&(Minisat::CRef)> smtrat::VarScheduler↩Base::getClause [protected], [inherited]`

15.577.4.8 getPolarity

`std::function<char(Minisat::Var)> smtrat::VarSchedulerBase::getPolarity [protected], [inherited]`

15.577.4.9 isAbstractedFormula

`std::function<bool(const FormulaT&)> smtrat::VarSchedulerBase↩::isAbstractedFormula [protected], [inherited]`

15.577.4.10 isBoolValueUndef

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::is↩BoolValueUndef [protected], [inherited]`

15.577.4.11 isDecisionVar

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::is↩DecisionVar [protected], [inherited]`

15.577.4.12 isTheoryAbstraction

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase↩::isTheoryAbstraction [protected], [inherited]`

15.577.4.13 reabstractLiteral

`std::function<const FormulaT&(Minisat::Lit)> smtrat::VarScheduler↩Base::reabstractLiteral [protected], [inherited]`

15.577.4.14 reabstractVariable

`std::function<const FormulaT&(Minisat::Var)> smtrat::VarScheduler↩Base::reabstractVariable [protected], [inherited]`

15.577.4.15 setPolarity

`std::function<void(Minisat::Var,bool)> smtrat::VarSchedulerBase::set↩Polarity [protected], [inherited]`

15.578 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)> 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

15.578.1 Detailed Description

template<TheoryGuidedDecisionHeuristicLevel theory_conflict_guided_decision_heuristic>
class smtrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic >

Scheduler for SMT, implementing theory guided heuristics.

15.578.2 Constructor & Destructor Documentation

15.578.2.1 VarSchedulerSMTTheoryGuided()

```
template<TheoryGuidedDecisionHeuristicLevel theory↩
_conflict_guided_decision_heuristic>
template<typename BaseModule >
smtrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic >::VarSchedulerSMTTheoryGuided
(
            BaseModule & baseModule )  [inline], [explicit]
```

15.578.3 Member Function Documentation

15.578.3.1 attachClause()

```
void smtrat::VarSchedulerBase::attachClause (
            Minisat::CRef  )  [inline], [inherited]
```

15.578.3.2 decreaseActivity()

```
template<TheoryGuidedDecisionHeuristicLevel theory_conflict_↩
guided_decision_heuristic>
void smtrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic >::decrease↩
Activity (
            Minisat::Var var )  [inline]
```

15.578.3.3 detachClause()

```
void smtrat::VarSchedulerBase::detachClause (
            Minisat::CRef  )  [inline], [inherited]
```

15.578.3.4 empty()

```
template<TheoryGuidedDecisionHeuristicLevel theory_conflict_guided_decision↩
_heuristic>
bool smtrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic >::empty (
)  [inline]
```

15.578.3.5 increaseActivity()

```
template<TheoryGuidedDecisionHeuristicLevel theory_conflict_guided_decision_heuristic>
void smtrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic >::increaseActivity (
            Minisat::Var var )  [inline]
```

15.578.3.6 insert()

```
template<TheoryGuidedDecisionHeuristicLevel theory_conflict_guided_decision_heuristic>
void smtrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic >::insert (
            Minisat::Var var )  [inline]
```

15.578.3.7 pop()

```
template<TheoryGuidedDecisionHeuristicLevel theory_conflict_guided_decision_heuristic>
Minisat::Lit smtrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic >::pop ( )  [inline]
```

15.578.3.8 print()

```
template<TheoryGuidedDecisionHeuristicLevel theory_conflict_guided_decision_heuristic>
void smtrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic >::print (
) const  [inline]
```

15.578.3.9 rebuild()

```
template<TheoryGuidedDecisionHeuristicLevel theory_conflict_guided_decision_heuristic>
void smtrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic >::rebuild
( )  [inline]
```

15.578.3.10 rebuildActivities()

```
template<TheoryGuidedDecisionHeuristicLevel theory_conflict_guided_decision_heuristic>
void smtrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic >::rebuildActivities ( )  [inline]
```

15.578.3.11 rebuildTheoryVars()

```
template<typename Constraints >
void smtrat::VarSchedulerBase::rebuildTheoryVars (
            const Constraints &  )  [inline], [inherited]
```

15.578.3.12 relocateClauses()

```
void smtrat::VarSchedulerBase::relocateClauses (
            std::function< void(Minisat::CRef &)>  )  [inline], [inherited]
```

15.578.4 Field Documentation

15.578.4.1 abstractLiteral

```
std::function<const Minisat::Lit(const FormulaT&)> smtrat::VarSchedulerBase::abstractLiteral  [protected], [inherited]
```

15.578.4.2 abstractVariable

`std::function< Minisat::Var(const FormulaT&)> smtrat::VarScheduler↩ Base::abstractVariable [protected], [inherited]`

15.578.4.3 currentlySatisfiedByBackend

`std::function<unsigned(const FormulaT&)> smtrat::Var↩ SchedulerBase::currentlySatisfiedByBackend [protected], [inherited]`

15.578.4.4 getActivity

`std::function<double(Minisat::Var)> smtrat::VarSchedulerBase::get↩ Activity [protected], [inherited]`

15.578.4.5 getBoolLitValue

`std::function<Minisat::lbool(Minisat::Lit)> smtrat::VarScheduler↩ Base::getBoolLitValue [protected], [inherited]`

15.578.4.6 getBoolVarValue

`std::function<Minisat::lbool(Minisat::Var)> smtrat::VarScheduler↩ Base::getBoolVarValue [protected], [inherited]`

15.578.4.7 getClause

`std::function<const Minisat::Clause&(Minisat::CRef)> smtrat::VarScheduler↩ Base::getClause [protected], [inherited]`

15.578.4.8 getPolarity

`std::function<char(Minisat::Var)> smtrat::VarSchedulerBase::getPolarity [protected], [inherited]`

15.578.4.9 isAbstractedFormula

`std::function<bool(const FormulaT&)> smtrat::VarSchedulerBase↩ ::isAbstractedFormula [protected], [inherited]`

15.578.4.10 isBoolValueUndef

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::is↩ BoolValueUndef [protected], [inherited]`

15.578.4.11 isDecisionVar

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::is↩ DecisionVar [protected], [inherited]`

15.578.4.12 isTheoryAbstraction

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase↩ ::isTheoryAbstraction [protected], [inherited]`

15.578.4.13 reabstractLiteral

`std::function<const FormulaT&(Minisat::Lit)> smtrat::VarScheduler↩ Base::reabstractLiteral [protected], [inherited]`

15.578.4.14 reabstractVariable `std::function<const FormulaT&(Minisat::Var)> smtrat::VarScheduler↩Base::reabstractVariable [protected], [inherited]`

15.578.4.15 setPolarity `std::function<void(Minisat::Var,bool)> smtrat::VarSchedulerBase::set↩Polarity [protected], [inherited]`

15.579 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)

15.579.1 Constructor & Destructor Documentation

15.579.1.1 vec() [1/4] `template<class T>`

```
Minisat::vec< T >::vec ( ) [inline]
```

15.579.1.2 vec() [2/4] `template<class T>`

```
Minisat::vec< T >::vec (
            int size ) [inline], [explicit]
```

15.579.1.3 vec() [3/4] `template<class T>`

```
Minisat::vec< T >::vec (
            int size,
            const T & pad ) [inline]
```

15.579.1.4 vec() [4/4] template<class T>

```
Minisat::vec< T >::vec (
            vec< T > && _toMove )  [inline]
```

15.579.1.5 ~vec() template<class T>

```
Minisat::vec< T >::~vec ( )  [inline]
```

15.579.2 Member Function Documentation

15.579.2.1 capacity() [1/2] template<class T >

```
void Minisat::vec< T >::capacity (
            int min_cap )
```

15.579.2.2 capacity() [2/2] template<class T>

```
int Minisat::vec< T >::capacity (
            void  ) const  [inline]
```

15.579.2.3 clear() template<class T >

```
void Minisat::vec< T >::clear (
            bool dealloc = false )
```

15.579.2.4 copyTo() template<class T>

```
void Minisat::vec< T >::copyTo (
            vec< T > & copy ) const  [inline]
```

15.579.2.5 growTo() [1/2] template<class T >

```
void Minisat::vec< T >::growTo (
            int size )
```

15.579.2.6 growTo() [2/2] template<class T>

```
void Minisat::vec< T >::growTo (
            int size,
            const T & pad )
```

15.579.2.7 last() [1/2] template<class T>

```
T& Minisat::vec< T >::last (
            void  )  [inline]
```

15.579.2.8 last() [2/2] template<class T>

```
const T& Minisat::vec< T >::last (
            void  ) const  [inline]
```

15.579.2.9 moveTo() template<class T>

```
void Minisat::vec< T >::moveTo (
            vec< T > & dest )  [inline]
```

15.579.2.10 operator T*() template<class T>

```
Minisat::vec< T >::operator T* (
            void  )  [inline]
```

15.579.2.11 operator=() template<class T>

```
vec<T>& Minisat::vec< T >::operator= (
            vec< T > && _toMove )  [inline]
```

15.579.2.12 operator[]() [1/2] template<class T>

```
T& Minisat::vec< T >::operator[] (
            int index )  [inline]
```

15.579.2.13 operator[]() [2/2] template<class T>

```
const T& Minisat::vec< T >::operator[] (
            int index ) const  [inline]
```

15.579.2.14 pop() template<class T>

```
void Minisat::vec< T >::pop (
            void  )  [inline]
```

15.579.2.15 push() [1/2] template<class T>

```
void Minisat::vec< T >::push (
            const T & elem )  [inline]
```

15.579.2.16 push() [2/2] template<class T>

```
void Minisat::vec< T >::push (
            void  )  [inline]
```

15.579.2.17 push_() template<class T>

```
void Minisat::vec< T >::push_ (
            const T & elem )  [inline]
```

15.579.2.18 shrink() template<class T>

```
void Minisat::vec< T >::shrink (
            int nelems )  [inline]
```

15.579.2.19 shrink_() template<class T>

```
void Minisat::vec< T >::shrink_ (
            int nelems )  [inline]
```

15.579.2.20 size() template<class T>

```
int Minisat::vec< T >::size (
            void  ) const  [inline]
```

15.580 smtrat::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

15.580.1 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.

15.580.2 Member Typedef Documentation

15.580.2.1 result_type template<typename Res>

```
typedef Res smtrat::parser::conversion::VectorVariantConverter< Res >::result_type
```

15.580.3 Member Function Documentation

15.580.3.1 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]
```

15.580.3.2 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]
```

15.581 smtrat::VSModule< Settings > Class Template Reference

```
#include <VSModule.h>
```

Public Types

- typedef Settings SettingsType
- enum LemmaType : unsigned { LemmaType::NORMAL = 0, LemmaType::PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- VSModule (const ModuleInput *, Conditionals &, Manager *const =NULL)
- ~VSModule ()
- 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 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 _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)

  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 &_branching↩Value) 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 _preferLeft↩Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩::vector< FormulaT >())

- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormula↩AsPremise=false)
- template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
  bool branchAt (const typename P::PolyType &_poly, bool _integral, const Rational &_value, bool _left↩CaseWeak=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.

15.581.1 Member Typedef Documentation

15.581.1.1 SettingsType `template<class Settings >`

```
typedef Settings smtrat::VSModule< Settings >::SettingsType
```

15.581.2 Member Enumeration Documentation

15.581.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

15.581.3 Constructor & Destructor Documentation

15.581.3.1 VSModule() `template<class Settings >`

```
smtrat::VSModule< Settings >::VSModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * const _manager = NULL )
```

15.581.3.2 ~VSModule() `template<class Settings >`

```
smtrat::VSModule< Settings >::~VSModule ( )
```

15.581.4 Member Function Documentation

15.581.4.1 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.

15.581.4.2 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 >.

15.581.4.3 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

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.

15.581.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

15.581.4.5 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.

15.581.4.6 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.

15.581.4.7 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.

15.581.4.8 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.

15.581.4.9 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.

15.581.4.10 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

_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.

15.581.4.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const  [inline],
[inherited]
```

Returns All satisfying assignments, if existent.

15.581.4.12 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.

15.581.4.13 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.

15.581.4.14 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.

15.581.4.15 branchAt() [1/6]

```
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]
```

15.581.4.16 branchAt() [2/6]

```
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]
```

15.581.4.17 branchAt() [3/6]

```
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]
```

15.581.4.18 branchAt() [4/6]

```
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

_polynomial	The variable to branch for.
_integral	A flag being true, if all variables in the polynomial to branch for are integral.
_value	The value to branch at.
_premise	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.
_leftCaseWeak	true, if a branching in the form of (or (<= p b) (> p b)) is desired. false, if a branching in the form of (or (< p b) (>= p b)) is desired.
_preferLeftCase	true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.
_useReceivedFormulaAsPremise	true, if the whole received formula should be used as premise

15.581.4.19 branchAt() [5/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
    const typename P::PolyType & _poly,
    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]
```

15.581.4.20 branchAt() [6/6]

```
template<typename P = Poly, carl::EnableIf< carl::needs_cache< P >> = carl::dummy>
bool smtrat::Module::branchAt (
    const typename P::PolyType & _poly,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

15.581.4.21 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

_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 received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in smtrat::PModule.

15.581.4.22 checkCore()

```
template<class Settings >
Answer smtrat::VSModule< Settings >::checkCore ( ) [virtual]
```

Checks the received formula for consistency.

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 SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

15.581.4.23 checkInfSubsetForMinimality()

```
void smtrat::Module::checkInfSubsetForMinimality (
    std::vector< FormulaSetT >::const_iterator _infsubset,
    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

_infsubset	The infeasible subset to check for minimality.
_filename	The name of the file to store the infeasible subsets in order to be able to check their infeasibility.
_maxSizeDifference	The maximal difference between the sizes of the subsets compared to the size of the infeasible subset.

15.581.4.24 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.

15.581.4.25 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

15.581.4.26 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

15.581.4.27 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

15.581.4.28 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

15.581.4.29 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

15.581.4.30 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

15.581.4.31 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

_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.

15.581.4.32 collectTheoryPropagations()

`void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

15.581.4.33 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns The constraints which the backends must be informed about.

15.581.4.34 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

15.581.4.35 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)

15.581.4.36 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

_constraint	The constraint to remove from internal data structures.

15.581.4.37 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

_constraint	The constraint to remove from internal data structures.

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

15.581.4.38 determine_smallest_origin()

```
size_t smtrat::Module::determine_smallest_origin (
    const std::vector< FormulaT > & origins ) const [protected], [inherited]
```

Parameters

origins	A vector of sets of origins.

Returns The index of the smallest (regarding the size of the sets) element of origins

15.581.4.39 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

_subformula	The sub-formula to remove from the passed formula.
_ignoreOrigins	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 >.

15.581.4.40 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

15.581.4.41 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

15.581.4.42 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.

15.581.4.43 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.

15.581.4.44 freeSplittingVariable()

```
static void smtrat::Module::freeSplittingVariable (
    const FormulaT & _splittingVariable ) [inline], [static], [inherited]
```

15.581.4.45 generateTrivialInfeasibleSubset()

```
void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]
```

Stores the trivial infeasible subset being the set of received formulas.

15.581.4.46 getBackendsAllModels()

```
void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]
```

Stores all models of a backend in the list of all models of this module.

15.581.4.47 getBackendsModel()

```
void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]
```

15.581.4.48 getInfeasibleSubsets() [1/2]

```
void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]
```

Copies the infeasible subsets of the passed formula.

15.581.4.49 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

_backend	The backend from which to obtain the infeasible subsets.

Returns The infeasible subsets the given backend provides.

15.581.4.50 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns Set of all formulas

15.581.4.51 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.

15.581.4.52 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.581.4.53 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

15.581.4.54 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

_formula	The position of a formula in the passed formulas.

Returns The origins of the passed formula at the given position.

15.581.4.55 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 reveived formula, if the Boolean is true.

Reimplemented in smtrat::PModule.

15.581.4.56 hasLemmas() bool smtrat::Module::hasLemmas ( ) [inline], [inherited]

Checks whether this module or any of its backends provides any lemmas.

15.581.4.57 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).

15.581.4.58 id() std::size_t smtrat::Module::id ( ) const [inline], [inherited]

Returns A unique ID to identify this module instance.

15.581.4.59 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.

15.581.4.60 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

15.581.4.61 informBackends()

```
void smtrat::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.

15.581.4.62 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

_constraint	The constraint to inform about.

Returns false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in smtrat::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::FPPModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, and smtrat::ICPModule< ICPSettings1 >.

15.581.4.63 informedConstraints()

```
const carl::FastSet<FormulaT>& smtrat::Module::informed↩
Constraints ( ) const [inline], [inherited]
```

Returns The position of the first constraint of which no backend has been informed about.

15.581.4.64 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::EQModule< Settings >, smtrat::EQModule< EQSettingsForPreprocessing >, smtrat::IntBlastModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::PBPPModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::BVModule< Settings >, smtrat::CurryModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, and smtrat::FPPModule< Settings >.

15.581.4.65 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

15.581.4.66 isLemmaLevel() `bool smtrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

level	Level to check.

15.581.4.67 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().

15.581.4.68 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.

15.581.4.69 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

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns The vector being the two given vectors merged.

15.581.4.70 model()

`const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns The assignment of the current satisfiable result, if existent.

15.581.4.71 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

_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.

15.581.4.72 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.

15.581.4.73 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

15.581.4.74 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
    const FormulaT & _origin ) const [protected], [inherited]
```

15.581.4.75 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula. TODO: disable this method

15.581.4.76 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns An iterator to the end of the passed formula.

15.581.4.77 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to the backends of this module.

15.581.4.78 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns A pointer to the formula passed to this module.

15.581.4.79 print()

```
void smtrat::Module::print (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

15.581.4.80 printAll()

```
template<class Settings >
void smtrat::VSModule< Settings >::printAll (
    const std::string & _init = "",
    std::ostream & _out = std::cout ) const
```

Prints the history to the output stream.

Parameters

_init	The beginning of each row.
_out	The output stream where the history should be printed.

15.581.4.81 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

_out	The stream to print the assignment on.

15.581.4.82 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

_init	The beginning of each row.
_out	The output stream where the answer should be printed.

15.581.4.83 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

_init	The beginning of each row.
_out	The output stream where the history should be printed.

15.581.4.84 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

15.581.4.85 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

_out	The stream to print the assignment on.

15.581.4.86 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

15.581.4.87 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

_init	The beginning of each row.
_out	The output stream where the history should be printed.

15.581.4.88 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

15.581.4.89 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.

15.581.4.90 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

15.581.4.91 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩
InfeasibleSubset (
            ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

15.581.4.92 receivedVariable()

```
bool smtrat::Module::receivedVariable (
            carl::Variable::Arg _var ) const [inline], [inherited]
```

15.581.4.93 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

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

15.581.4.94 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.

15.581.4.95 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

15.581.4.96 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]
```

15.581.4.97 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]
```

Returns A reference to the formula passed to the backends of this module.

15.581.4.98 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]
```

Returns A reference to the formula passed to this module.

15.581.4.99 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]
```

Reimplemented in smtrat::PModule.

15.581.4.100 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

_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 in smtrat::PModule.

15.581.4.101 setId()

`void smtrat::Module::setId ( std::size_t _id ) [inline], [inherited]`

Sets this modules unique ID to identify itself.

Parameters

↩ *_id*	The id to set this module's id to.

15.581.4.102 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

_threadPriority	The priority to set this module's thread priority to.

15.581.4.103 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.

15.581.4.104 splitUnequalConstraint()

`void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint p!=0.

```
(p!=0 <-> (p<0 or p>0))
```

and not(p<0 and p>0)

Parameters

_unequalConstraint	A constraint having the relation symbol !=.

15.581.4.105 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns The priority of this module to get a thread for running its check procedure.

15.581.4.106 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 >.

15.581.4.107 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

15.581.4.108 updateModel() `template<class Settings > void smtrat::VSModule< Settings >::updateModel ( ) const [virtual]`

Updates the model, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented from smtrat::Module.

15.581.4.109 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).

15.581.5 Field Documentation

15.581.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

15.581.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

15.581.5.3 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.

15.581.5.4 mConstraintsToInform

`carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

15.581.5.5 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

15.581.5.6 mFirstPosInLastBranches

`std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

15.581.5.7 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.

15.581.5.8 mFirstUncheckedReceivedSubformula

`ModuleInput::const_iterator smtrat::Module::m↩FirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

15.581.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

15.581.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

15.581.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

15.581.5.12 mInformedConstraints `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

15.581.5.13 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.

15.581.5.14 mLemmas `std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

15.581.5.15 mModel `Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

15.581.5.16 mModelComputed `bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

15.581.5.17 mNumOfBranchVarsToStore `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

15.581.5.18 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.

15.581.5.19 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

15.581.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

15.581.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

15.581.5.22 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.

15.581.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

15.581.5.24 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).

15.581.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

15.582 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)

15.582.1 Detailed Description [Minisat related code]

15.582.2 Constructor & Destructor Documentation

15.582.2.1 Watcher()

```
Minisat::Watcher::Watcher (
    Minisat::CRef cr,
    Minisat::Lit p ) [inline]
```

[Minisat related code]

15.582.3 Member Function Documentation

15.582.3.1 operator"!=()

```
bool Minisat::Watcher::operator!= (
    const Watcher & w ) const [inline]
```

[Minisat related code]

15.582.3.2 operator==()

```
bool Minisat::Watcher::operator== (
    const Watcher & w ) const [inline]
```

[Minisat related code]

15.582.4 Friends And Related Function Documentation

15.582.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const Watcher & w ) [friend]
```

15.582.5 Field Documentation

15.582.5.1 blocker

```
Minisat::Lit Minisat::Watcher::blocker
```

[Minisat related code]

15.582.5.2 cref

```
Minisat::CRef Minisat::Watcher::cref
```

[Minisat related code]

15.583 smtrat::ICPModule< Settings >::weights Struct Reference

```
#include <ICPModule.h>
```

Data Fields

- std::list< linearVariable * > origins
- double weight

15.583.1 Field Documentation

15.583.1.1 origins

```
template<class Settings>
std::list<linearVariable*> smtrat::ICPModule< Settings >::weights::origins
```

15.583.1.2 weight

```
template<class Settings>
double smtrat::ICPModule< Settings >::weights::weight
```

15.584 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.

15.584.1 Detailed Description

Writes results to a xml file.

15.584.2 Constructor & Destructor Documentation

15.584.2.1 XMLWriter()

```
benchmax::XMLWriter::XMLWriter (
    const std::string & filename ) [inline]
```

Open file for writing.

15.584.3 Member Function Documentation

15.584.3.1 write()

```
template<typename Results >
void benchmax::XMLWriter::write (
    const Jobs & jobs,
    const Results & results ) [inline]
```

Write results to file.

15.585 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.

15.585.1 Detailed Description

Tool wrapper for Z3 for SMT-LIB.

15.585.2 Constructor & Destructor Documentation

15.585.2.1 Z3()

```
benchmax::Z3::Z3 (
    const fs::path & binary,
    const std::string & arguments ) [inline]
```

New Z3 tool.

15.585.3 Member Function Documentation

15.585.3.1 additionalResults()

```
virtual void benchmax::Z3::additionalResults (
    const fs::path & ,
    BenchmarkResult & result ) const [inline], [override], [virtual]
```

Parse answer string from stdout.

Reimplemented from benchmax::Tool.

15.585.3.2 attributeHash()

```
std::size_t benchmax::Tool::attributeHash ( ) const [inline], [inherited]
```

Hash of the attributes.

15.585.3.3 attributes()

```
const std::map<std::string,std::string>& benchmax::Tool::attributes ( )
const [inline], [inherited]
```

A set of attributes, for example compilation options.

15.585.3.4 binary()

```
fs::path benchmax::Tool::binary ( ) const [inline], [inherited]
```

Full path to the binary.

15.585.3.5 canHandle()

```
virtual bool benchmax::Z3::canHandle (
    const fs::path & path ) const [inline], [override], [virtual]
```

Only handle .smt2 files.

Reimplemented from benchmax::Tool.

15.585.3.6 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.

15.585.3.7 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.

15.585.3.8 name()

```
std::string benchmax::Tool::name ( ) const [inline], [inherited]
```

Common name of this tool.

15.585.3.9 parseCommandline()

```
virtual std::optional<std::string> benchmax::Tool::parse↩
Commandline (
    const std::string & cmdline ) const [inline], [virtual], [inherited]
```

Compose commandline for this tool and the given input file.

15.585.3.10 resolveDependencies()

```
std::vector<std::string> benchmax::Tool::resolveDependencies
( ) const [inline], [inherited]
```

Get dependencies of binary required to run it (via ldd)

15.585.4 Field Documentation

15.585.4.1 mArguments

```
std::string benchmax::Tool::mArguments [protected], [inherited]
```

Command line arguments that should be passed to the binary.

15.585.4.2 mAttributes

```
std::map<std::string,std::string> benchmax::Tool::mAttributes [protected],
[inherited]
```

Attributes of the tool obtained by introspection of the binary.

15.585.4.3 mBinary

```
fs::path benchmax::Tool::mBinary [protected], [inherited]
```

Path to the binary.

15.585.4.4 mName

```
std::string benchmax::Tool::mName [protected], [inherited]
```

(Non-unique) identifier for the tool.

References

[1] Cristina Borralleras, Salvador Lucas, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. Sat modulo linear arithmetic for solving polynomial constraints. *Journal of Automated Reasoning*, 48(1):107–131, Jan 2012.

[2] Martin Bromberger and Christoph Weidenbach. Fast cube tests for lia constraint solving. In Nicola Olivetti and Ashish Tiwari, editors, *Automated Reasoning*, pages 116–132, Cham, 2016. Springer International Publishing.

[3] Christopher W. Brown. Improved projection for cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 32(5):447 – 465, 2001.

[4] Alessandro Cimatti, Alberto Griggio, Ahmed Irfan, Marco Roveri, and Roberto Sebastiani. Satisfiability modulo transcendental functions via incremental linearization. In Leonardo de Moura, editor, *Automated Deduction – CADE 26*, pages 95–113, Cham, 2017. Springer International Publishing.

[5] G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages*, volume 33 of *LNCS*, pages 134–183. Springer, 1975.

[6] F. Corzilius and E. Ábrahám. Virtual substitution for SMT solving. In 18th Int. Symp. on Fundamentals of Computation Theory, 2011.

[7] Isil Dillig, Thomas Dillig, and Alex Aiken. Cuts from proofs: a complete and practical technique for solving linear inequalities over integers. *Formal Methods in System Design*, 39(3):246–260, 2011.

[8] B. Dutertre and L. de Moura. A fast linear-arithmetic solver for DPLL(T). In *Proc. of CAV'06*, volume 4144 of *LNCS*, pages 81–94. Springer, 2006.

[9] Pascal Fontaine, Mizuhito Ogawa, Thomas Sturm, and Xuan Tung Vu. Subtropical satisfiability. In *International Symposium on Frontiers of Combining Systems*, pages 189–206. Springer, 2017.

[10] Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, and Harald Zankl. Sat solving for termination analysis with polynomial interpretations. In João Marques-Silva and Karem A. Sakallah, editors, *Theory and Applications of Satisfiability Testing – SAT 2007*, pages 340–354, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[11] S. Gao, M. K. Ganai, F. Ivancic, A. Gupta, S. Sankaranarayanan, and E. M. Clarke. Integrating ICP and LRA solvers for deciding nonlinear real arithmetic problems. In *Proc. of FMCAD'10*, pages 81–89. IEEE, 2010.

[12] Alberto Griggio. A practical approach to satisfiability modulo linear integer arithmetic. *Journal on Satisfiability, Boolean Modeling and Computation*, 8:1–27, 2012.

[13] Marta Grobelna. Solving pseudo-boolean constraints, 2017.

[14] Stefan Herbort and Dietmar Ratz. Improving the efficiency of a nonlinear-system-solver using a componentwise newton method. 1997.

[15] H. Hong. An improvement of the projection operator in cylindrical algebraic decomposition. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, ISSAC '90, pages 261–264, New York, NY, USA, 1990. ACM.

[16] Ahmed Irfan. *Incremental linearization for satisfiability and verification modulo nonlinear arithmetic and transcendental functions*. PhD thesis, University of Trento, 2018.

[17] S. Junges, U. Loup, F. Corzilius, and E. Ábrahám. On Gröbner bases in the context of satisfiability-modulo-theories solving over the real numbers. Technical Report AIB-2013-08, RWTH Aachen University, 2013.

[18] Gereon Kremer, Florian Corzilius, and Erika Ábrahám. A Generalised Branch-and-Bound Approach and Its Application in SAT Modulo Nonlinear Integer Arithmetic. In *Proc. of CASC'16*, pages 315–335. Springer, 2016.

[19] Gereon Kremer and Erika Ábrahám. Fully incremental cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 2019.

[20] Andreas Krüger. Bitvectors in smt-rat and their application to integer arithmetics, 2015.

[21] Daniel Lazard. *An Improved Projection for Cylindrical Algebraic Decomposition*, chapter 29, pages 467–476. Springer, 1994.

[22] U. Loup, K. Scheibler, F. Corzilius, Erika E. Ábrahám, and Bernd Becker. A symbiosis of interval constraint propagation and cylindrical algebraic decomposition. In *Proc. of CADE-24*, volume 7898 of *LNCS*, pages 193–207. Springer, 2013.

[23] Scott McCallum. *An improved projection operation for cylindrical algebraic decomposition*, pages 277–278. Springer Berlin Heidelberg, Berlin, Heidelberg, 1985.

[24] Aina Niemetz and Armin Biere. ddsmt: A delta debugger for the smt-lib v2 format. In *SMT Workshop 2013 11th International Workshop on Satisfiability Modulo Theories*, 2013.

[25] Robert Nieuwenhuis and Albert Oliveras. Fast congruence closure and extensions. *Information and Computation*, 205(4):557 – 580, 2007. Special Issue: 16th International Conference on Rewriting Techniques and Applications.

[26] Aklima Zaman. Incremental linearization for sat modulo real arithmetic solving, 2019.

[27] Ömer Sali. Linearization techniques for nonlinear arithmetic problems in smt, 2018.