Your title here

Generated by Doxygen 1.9.1

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

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

0.7.5.3 Backends . . . 24
0.7.5.4 Troubleshooting . . . 25
0.7.5.5 Benchmax python utility . . . 25
0.8 Developers information . . . 26
0.8.1 Code style . . . 27
0.8.2 Documentation . . . 27
0.8.2.1 Code comments . . . 27
0.8.2.2 Writing out-of-source documentation . . . 28
0.8.3 Settings . . . 28
0.8.4 Logging . . . 29
0.8.5 Statistics and timing . . . 30
0.8.6 Testing . . . 31
0.8.7 Validation . . . 31
0.8.8 Checkpoints . . . 32
0.8.9 Finding and Reporting Bugs . . . 33
0.9 README . . . 33
0.10 NewCoveringModule #and . . . 34
0.10.1 Introduction . . . 34
0.10.2 Usage . . . 34
0.10.3 Efficiency . . . 34
0.11 Todo List . . . 34
0.12 Hierarchical Index . . . 35
0.12.1 Class Hierarchy . . . 35
0.13 Data Structure Index . . . 47
0.13.1 Data Structures . . . 47
0.14 Namespace Documentation . . . 60
0.14.1 benchmax Namespace Reference . . . 60
0.14.1.1 Typedef Documentation . . . 62
0.14.1.2 Function Documentation . . . 62
0.14.2 benchmax::settings Namespace Reference . . . 65
0.14.2.1 Function Documentation . . . 66
0.14.3 benchmax::slurm Namespace Reference . . . 67
0.14.3.1 Function Documentation . . . 67
0.14.4 benchmax::ssh Namespace Reference . . . 69
0.14.5 carl Namespace Reference . . . 69
0.14.6 Minisat Namespace Reference . . . 69
0.14.6.1 Typedef Documentation . . . 71
0.14.6.2 Function Documentation . . . 71
0.14.6.3 Variable Documentation . . . 75
0.14.7 smtrat Namespace Reference . . . 75
0.14.7.1 Detailed Description . . . 83
0.14.7.2 Typedef Documentation . . . 84

0.14.7.3 Enumeration Type Documentation . . . 87
0.14.7.4 Function Documentation . . . 88
0.14.7.5 Variable Documentation . . . 96
0.14.8 smtrat::analyzer Namespace Reference . . . 96
0.14.8.1 Function Documentation . . . 96
0.14.9 smtrat::cad Namespace Reference . . . 97
0.14.9.1 Typedef Documentation . . . 100
0.14.9.2 Enumeration Type Documentation . . . 100
0.14.9.3 Function Documentation . . . 102
0.14.10 smtrat::cad::debug Namespace Reference . . . 104
0.14.11 smtrat::cad::full Namespace Reference . . . 104
0.14.11.1 Typedef Documentation . . . 104
0.14.12 smtrat::cad::full_ec Namespace Reference . . . 104
0.14.12.1 Typedef Documentation . . . 105
0.14.13 smtrat::cad::preprocessor Namespace Reference . . . 105
0.14.13.1 Function Documentation . . . 105
0.14.14 smtrat::cad::projection Namespace Reference . . . 105
0.14.14.1 Function Documentation . . . 106
0.14.15 smtrat::cad::projection::brown Namespace Reference . . . 107
0.14.15.1 Detailed Description . . . 107
0.14.15.2 Function Documentation . . . 108
0.14.16 smtrat::cad::projection::collins Namespace Reference . . . 108
0.14.16.1 Detailed Description . . . 108
0.14.16.2 Function Documentation . . . 108
0.14.17 smtrat::cad::projection::hong Namespace Reference . . . 109
0.14.17.1 Detailed Description . . . 109
0.14.17.2 Function Documentation . . . 109
0.14.18 smtrat::cad::projection::lazard Namespace Reference . . . 109
0.14.18.1 Detailed Description . . . 110
0.14.18.2 Function Documentation . . . 110
0.14.19 smtrat::cad::projection::mccallum Namespace Reference . . . 110
0.14.19.1 Detailed Description . . . 110
0.14.19.2 Function Documentation . . . 110
0.14.20 smtrat::cad::projection::mccallum_partial Namespace Reference . . . 111
0.14.20.1 Detailed Description . . . 111
0.14.20.2 Function Documentation . . . 111
0.14.21 smtrat::cad::projection_compare Namespace Reference . . . 111
0.14.21.1 Typedef Documentation . . . 112
0.14.21.2 Function Documentation . . . 112
0.14.22 smtrat::cad::sample_compare Namespace Reference . . . 113
0.14.22.1 Detailed Description . . . 114
0.14.22.2 Typedef Documentation . . . 114

0.14.22.3 Function Documentation . . . 114
0.14.23 smtrat::cad::variable_ordering Namespace Reference . . . 115
0.14.23.1 Function Documentation . . . 115
0.14.24 smtrat::cadcells Namespace Reference . . . 115
0.14.24.1 Detailed Description . . . 116
0.14.24.2 Typedef Documentation . . . 116
0.14.24.3 Variable Documentation . . . 117
0.14.25 smtrat::cadcells::algorithms Namespace Reference . . . 117
0.14.25.1 Detailed Description . . . 118
0.14.25.2 Function Documentation . . . 118
0.14.26 smtrat::cadcells::datastructures Namespace Reference . . . 119
0.14.26.1 Detailed Description . . . 122
0.14.26.2 Typedef Documentation . . . 123
0.14.26.3 Function Documentation . . . 123
0.14.27 smtrat::cadcells::datastructures::detail Namespace Reference . . . 128
0.14.28 smtrat::cadcells::helper Namespace Reference . . . 128
0.14.28.1 Function Documentation . . . 129
0.14.29 smtrat::cadcells::operators Namespace Reference . . . 129
0.14.29.1 Detailed Description . . . 130
0.14.29.2 Enumeration Type Documentation . . . 131
0.14.29.3 Function Documentation . . . 131
0.14.29.4 Variable Documentation . . . 133
0.14.30 smtrat::cadcells::operators::mccallum_filtered_impl Namespace Reference . . . 134
0.14.30.1 Function Documentation . . . 134
0.14.31 smtrat::cadcells::operators::properties Namespace Reference . . . 134
0.14.31.1 Detailed Description . . . 135
0.14.31.2 Function Documentation . . . 135
0.14.32 smtrat::cadcells::operators::rules Namespace Reference . . . 139
0.14.32.1 Detailed Description . . . 141
0.14.32.2 Function Documentation . . . 141
0.14.33 smtrat::cadcells::operators::rules::additional_root_outside_util Namespace Reference . . . 145
0.14.33.1 Function Documentation . . . 145
0.14.34 smtrat::cadcells::operators::rules::compound_util Namespace Reference . . . 145
0.14.34.1 Function Documentation . . . 146
0.14.35 smtrat::cadcells::operators::rules::filter_util Namespace Reference . . . 146
0.14.35.1 Enumeration Type Documentation . . . 147
0.14.35.2 Function Documentation . . . 147
0.14.36 smtrat::cadcells::operators::rules::ordering_util Namespace Reference . . . 148
0.14.36.1 Typedef Documentation . . . 148
0.14.36.2 Function Documentation . . . 148
0.14.37 smtrat::cadcells::representation Namespace Reference . . . 148
0.14.37.1 Detailed Description . . . 149

0.14.37.2 Typedef Documentation . . . 149
0.14.37.3 Enumeration Type Documentation . . . 149
0.14.37.4 Function Documentation . . . 150
0.14.37.5 Variable Documentation . . . 151
0.14.38 smtrat::cadcells::representation::approximation Namespace Reference . . . 151
0.14.38.1 Typedef Documentation . . . 152
0.14.38.2 Enumeration Type Documentation . . . 152
0.14.38.3 Function Documentation . . . 153
0.14.39 smtrat::cadcells::representation::util Namespace Reference . . . 155
0.14.39.1 Typedef Documentation . . . 156
0.14.39.2 Function Documentation . . . 156
0.14.40 smtrat::compile_information Namespace Reference . . . 158
0.14.40.1 Detailed Description . . . 158
0.14.40.2 Variable Documentation . . . 158
0.14.41 smtrat::covering_ng Namespace Reference . . . 159
0.14.41.1 Typedef Documentation . . . 160
0.14.41.2 Enumeration Type Documentation . . . 160
0.14.41.3 Function Documentation . . . 160
0.14.42 smtrat::covering_ng::formula Namespace Reference . . . 161
0.14.42.1 Enumeration Type Documentation . . . 162
0.14.42.2 Function Documentation . . . 162
0.14.43 smtrat::covering_ng::formula::exhaustive_implicants_helper Namespace Reference . . . 162
0.14.43.1 Function Documentation . . . 162
0.14.44 smtrat::covering_ng::formula::minimal_helper Namespace Reference . . . 162
0.14.44.1 Detailed Description . . . 163
0.14.44.2 Function Documentation . . . 163
0.14.45 smtrat::covering_ng::formula::node_ds Namespace Reference . . . 163
0.14.45.1 Function Documentation . . . 163
0.14.46 smtrat::covering_ng::formula::node_ds::complexity Namespace Reference . . . 164
0.14.46.1 Function Documentation . . . 164
0.14.47 smtrat::execution Namespace Reference . . . 165
0.14.47.1 Enumeration Type Documentation . . . 165
0.14.48 smtrat::expression Namespace Reference . . . 165
0.14.48.1 Typedef Documentation . . . 166
0.14.48.2 Enumeration Type Documentation . . . 166
0.14.48.3 Function Documentation . . . 167
0.14.49 smtrat::expression::simplifier Namespace Reference . . . 168
0.14.49.1 Typedef Documentation . . . 169
0.14.50 smtrat::groebner Namespace Reference . . . 169
0.14.50.1 Typedef Documentation . . . 169
0.14.50.2 Function Documentation . . . 169
0.14.51 smtrat::helper Namespace Reference . . . 169

0.14.51.1 Function Documentation . . . 169
0.14.52 smtrat::icp Namespace Reference . . . 169
0.14.52.1 Typedef Documentation . . . 170
0.14.52.2 Enumeration Type Documentation . . . 170
0.14.52.3 Function Documentation . . . 170
0.14.53 smtrat::lra Namespace Reference . . . 172
0.14.53.1 Typedef Documentation . . . 173
0.14.53.2 Function Documentation . . . 173
0.14.53.3 Variable Documentation . . . 178
0.14.54 smtrat::lve Namespace Reference . . . 178
0.14.54.1 Function Documentation . . . 178
0.14.55 smtrat::maxsmt Namespace Reference . . . 179
0.14.55.1 Detailed Description . . . 179
0.14.56 smtrat::mcsat Namespace Reference . . . 179
0.14.56.1 Typedef Documentation . . . 180
0.14.56.2 Enumeration Type Documentation . . . 181
0.14.56.3 Function Documentation . . . 181
0.14.57 smtrat::mcsat::arithmetic Namespace Reference . . . 183
0.14.57.1 Function Documentation . . . 183
0.14.58 smtrat::mcsat::constraint_type Namespace Reference . . . 183
0.14.58.1 Function Documentation . . . 183
0.14.59 smtrat::mcsat::fm Namespace Reference . . . 184
0.14.59.1 Detailed Description . . . 185
0.14.59.2 Function Documentation . . . 185
0.14.60 smtrat::mcsat::icp Namespace Reference . . . 185
0.14.61 smtrat::mcsat::nlsat Namespace Reference . . . 185
0.14.62 smtrat::mcsat::nlsat::helper Namespace Reference . . . 186
0.14.62.1 Function Documentation . . . 186
0.14.63 smtrat::mcsat::onecell Namespace Reference . . . 187
0.14.63.1 Typedef Documentation . . . 187
0.14.63.2 Function Documentation . . . 187
0.14.63.3 Variable Documentation . . . 188
0.14.64 smtrat::mcsat::onecellcad Namespace Reference . . . 188
0.14.64.1 Typedef Documentation . . . 189
0.14.64.2 Enumeration Type Documentation . . . 190
0.14.64.3 Function Documentation . . . 190
0.14.65 smtrat::mcsat::onecellcad::levelwise Namespace Reference . . . 194
0.14.66 smtrat::mcsat::onecellcad::recursive Namespace Reference . . . 195
0.14.66.1 Detailed Description . . . 195
0.14.66.2 Enumeration Type Documentation . . . 196
0.14.66.3 Function Documentation . . . 196
0.14.66.4 Variable Documentation . . . 196

0.14.67 smtrat::mcsat::smtaf Namespace Reference . . . 196
0.14.67.1 Typedef Documentation . . . 197
0.14.67.2 Function Documentation . . . 197
0.14.68 smtrat::mcsat::variableordering Namespace Reference . . . 197
0.14.68.1 Function Documentation . . . 197
0.14.69 smtrat::mcsat::variableordering::detail Namespace Reference . . . 198
0.14.69.1 Function Documentation . . . 199
0.14.70 smtrat::mcsat::vs Namespace Reference . . . 200
0.14.71 smtrat::mcsat::vs::helper Namespace Reference . . . 200
0.14.71.1 Function Documentation . . . 200
0.14.72 smtrat::onecellcad Namespace Reference . . . 201
0.14.73 smtrat::onecellcad::recursive Namespace Reference . . . 201
0.14.73.1 Typedef Documentation . . . 202
0.14.73.2 Function Documentation . . . 202
0.14.74 smtrat::options_detail Namespace Reference . . . 204
0.14.74.1 Function Documentation . . . 204
0.14.75 smtrat::parseformula Namespace Reference . . . 204
0.14.76 smtrat::parser Namespace Reference . . . 204
0.14.76.1 Typedef Documentation . . . 206
0.14.76.2 Enumeration Type Documentation . . . 207
0.14.76.3 Function Documentation . . . 207
0.14.77 smtrat::parser::arithmetic Namespace Reference . . . 208
0.14.77.1 Typedef Documentation . . . 208
0.14.77.2 Function Documentation . . . 208
0.14.78 smtrat::parser::conversion Namespace Reference . . . 208
0.14.79 smtrat::parser::core Namespace Reference . . . 209
0.14.79.1 Function Documentation . . . 209
0.14.80 smtrat::parser::types Namespace Reference . . . 209
0.14.80.1 Typedef Documentation . . . 210
0.14.81 smtrat::parser::uninterpreted Namespace Reference . . . 211
0.14.81.1 Function Documentation . . . 211
0.14.82 smtrat::qe Namespace Reference . . . 211
0.14.82.1 Typedef Documentation . . . 212
0.14.82.2 Enumeration Type Documentation . . . 212
0.14.82.3 Function Documentation . . . 212
0.14.83 smtrat::qe::cad Namespace Reference . . . 212
0.14.83.1 Function Documentation . . . 213
0.14.84 smtrat::qe::fm Namespace Reference . . . 213
0.14.84.1 Function Documentation . . . 213
0.14.85 smtrat::resource Namespace Reference . . . 213
0.14.85.1 Function Documentation . . . 213
0.14.86 smtrat::sat Namespace Reference . . . 214

0.14.87 smtrat::sat::detail Namespace Reference . . . 214
0.14.87.1 Function Documentation . . . 214
0.14.88 smtrat::settings Namespace Reference . . . 215
0.14.89 smtrat::statistics Namespace Reference . . . 215
0.14.89.1 Function Documentation . . . 215
0.14.90 smtrat::subtropical Namespace Reference . . . 215
0.14.90.1 Detailed Description . . . 216
0.14.90.2 Enumeration Type Documentation . . . 216
0.14.90.3 Function Documentation . . . 216
0.14.91 smtrat::unsatcore Namespace Reference . . . 217
0.14.91.1 Detailed Description . . . 217
0.14.92 smtrat::validation Namespace Reference . . . 217
0.14.92.1 Enumeration Type Documentation . . . 218
0.14.92.2 Function Documentation . . . 218
0.14.93 smtrat::vb Namespace Reference . . . 219
0.14.93.1 Function Documentation . . . 219
0.14.94 smtrat::vs Namespace Reference . . . 219
0.14.94.1 Typedef Documentation . . . 222
0.14.94.2 Function Documentation . . . 222
0.15 Data Structure Documentation . . . 232
0.15.1 smtrat::AbstractModuleFactory Struct Reference . . . 232
0.15.1.1 Constructor & Destructor Documentation . . . 232
0.15.1.2 Member Function Documentation . . . 232
0.15.2 smtrat::parser::AbstractTheory Struct Reference . . . 232
0.15.2.1 Detailed Description . . . 233
0.15.2.2 Constructor & Destructor Documentation . . . 233
0.15.2.3 Member Function Documentation . . . 234
0.15.2.4 Field Documentation . . . 235
0.15.3 smtrat::cad::sample_compare::absvalue Struct Reference . . . 235
0.15.4 smtrat::analyzer::AnalysisSettings Struct Reference . . . 235
0.15.4.1 Field Documentation . . . 235
0.15.5 smtrat::analyzer::AnalyzerStatistics Struct Reference . . . 236
0.15.5.1 Member Function Documentation . . . 236
0.15.6 smtrat::covering_ng::formula::node_ds::AND Struct Reference . . . 236
0.15.6.1 Field Documentation . . . 236
0.15.7 smtrat::AnnotatedBVTerm Class Reference . . . 236
0.15.7.1 Constructor & Destructor Documentation . . . 236
0.15.7.2 Member Function Documentation . . . 237
0.15.7.3 Friends And Related Function Documentation . . . 237
0.15.8 smtrat::cadcells::representation::approximation::ApxCriteria Class Reference . . . 237
0.15.8.1 Member Function Documentation . . . 237
0.15.9 smtrat::cadcells::representation::approximation::ApxSettings Struct Reference . . . 238

0.15.9.1 Field Documentation . . . 239
0.15.10 benchmax::slurm::ArchiveProperties Struct Reference . . . 240
0.15.10.1 Detailed Description . . . 241
0.15.10.2 Field Documentation . . . 241
0.15.11 smtrat::parser::ArithmeticTheory Struct Reference . . . 241
0.15.11.1 Detailed Description . . . 242
0.15.11.2 Constructor & Destructor Documentation . . . 242
0.15.11.3 Member Function Documentation . . . 242
0.15.11.4 Field Documentation . . . 243
0.15.12 smtrat::parser::types::ArithmeticTheory Struct Reference . . . 244
0.15.12.1 Detailed Description . . . 244
0.15.12.2 Member Typedef Documentation . . . 244
0.15.13 smtrat::execution::Assertion Struct Reference . . . 244
0.15.13.1 Field Documentation . . . 244
0.15.14 smtrat::cad::preprocessor::AssignmentCollector Class Reference . . . 244
0.15.14.1 Member Typedef Documentation . . . 245
0.15.14.2 Constructor & Destructor Documentation . . . 245
0.15.14.3 Member Function Documentation . . . 245
0.15.15 smtrat::mcsat::arithmetic::AssignmentFinder Struct Reference . . . 246
0.15.15.1 Member Function Documentation . . . 246
0.15.16 smtrat::mcsat::smtaf::AssignmentFinder< Settings > Struct Template Reference . . . 246
0.15.16.1 Member Function Documentation . . . 246
0.15.17 smtrat::mcsat::arithmetic::AssignmentFinder_ctx Class Reference . . . 246
0.15.17.1 Constructor & Destructor Documentation . . . 247
0.15.17.2 Member Function Documentation . . . 247
0.15.18 smtrat::mcsat::arithmetic::AssignmentFinder_detail Class Reference . . . 247
0.15.18.1 Constructor & Destructor Documentation . . . 247
0.15.18.2 Member Function Documentation . . . 248
0.15.19 smtrat::mcsat::smtaf::AssignmentFinder_SMT Class Reference . . . 248
0.15.19.1 Constructor & Destructor Documentation . . . 248
0.15.19.2 Member Function Documentation . . . 248
0.15.20 smtrat::cadcells::datastructures::detail::AssignmentProperties Struct Reference . . . 249
0.15.20.1 Field Documentation . . . 249
0.15.21 smtrat::parser::Attribute Class Reference . . . 249
0.15.21.1 Detailed Description . . . 250
0.15.21.2 Member Typedef Documentation . . . 250
0.15.21.3 Constructor & Destructor Documentation . . . 250
0.15.21.4 Member Function Documentation . . . 250
0.15.21.5 Field Documentation . . . 250
0.15.22 smtrat::parser::AttributeParser Struct Reference . . . 250
0.15.22.1 Constructor & Destructor Documentation . . . 251
0.15.22.2 Field Documentation . . . 251

0.15.23 smtrat::parser::AttributeValueParser Struct Reference . . . 251
0.15.23.1 Member Typedef Documentation . . . 251
0.15.23.2 Constructor & Destructor Documentation . . . 251
0.15.23.3 Field Documentation . . . 252
0.15.24 smtrat::AxiomFactory Class Reference . . . 252
0.15.24.1 Member Enumeration Documentation . . . 252
0.15.24.2 Member Function Documentation . . . 252
0.15.25 benchmax::Backend Class Reference . . . 253
0.15.25.1 Detailed Description . . . 253
0.15.25.2 Constructor & Destructor Documentation . . . 253
0.15.25.3 Member Function Documentation . . . 253
0.15.25.4 Field Documentation . . . 255
0.15.26 smtrat::Backend< Settings > Class Template Reference . . . 255
0.15.26.1 Constructor & Destructor Documentation . . . 255
0.15.26.2 Member Function Documentation . . . 255
0.15.27 smtrat::BackendLink Class Reference . . . 257
0.15.27.1 Constructor & Destructor Documentation . . . 258
0.15.27.2 Member Function Documentation . . . 258
0.15.28 smtrat::BackendSynchronisation Class Reference . . . 258
0.15.28.1 Constructor & Destructor Documentation . . . 258
0.15.28.2 Member Function Documentation . . . 259
0.15.29 smtrat::Backtrackable< UnionFind > Struct Template Reference . . . 259
0.15.29.1 Member Typedef Documentation . . . 259
0.15.29.2 Constructor & Destructor Documentation . . . 259
0.15.29.3 Member Function Documentation . . . 260
0.15.29.4 Field Documentation . . . 260
0.15.30 smtrat::mcsat::onecell::Base Struct Reference . . . 260
0.15.30.1 Field Documentation . . . 261
0.15.31 smtrat::cadcells::datastructures::BaseDerivation< Properties > Class Template Reference 261
0.15.31.1 Detailed Description . . . 261
0.15.31.2 Constructor & Destructor Documentation . . . 261
0.15.31.3 Member Function Documentation . . . 262
0.15.31.4 Friends And Related Function Documentation . . . 263
0.15.32 smtrat::cad::BaseProjection< Settings > Class Template Reference . . . 263
0.15.32.1 Member Typedef Documentation . . . 264
0.15.32.2 Constructor & Destructor Documentation . . . 264
0.15.32.3 Member Function Documentation . . . 264
0.15.32.4 Field Documentation . . . 268
0.15.33 smtrat::cad::BaseSettings Struct Reference . . . 268
0.15.33.1 Field Documentation . . . 269
0.15.34 smtrat::expression::simplifier::BaseSimplifier Struct Reference . . . 269
0.15.34.1 Member Function Documentation . . . 270

0.15.35 smtrat::cad::Origin::BaseType Struct Reference . . . 271
0.15.35.1 Constructor & Destructor Documentation . . . 271
0.15.35.2 Member Function Documentation . . . 272
0.15.35.3 Friends And Related Function Documentation . . . 272
0.15.35.4 Field Documentation . . . 272
0.15.36 smtrat::mcsat::onecell::BCApproximationSettings Struct Reference . . . 273
0.15.36.1 Field Documentation . . . 273
0.15.37 smtrat::mcsat::onecell::BCFilteredAllSelectiveSettings Struct Reference . . . 273
0.15.37.1 Field Documentation . . . 273
0.15.38 smtrat::mcsat::onecell::BCFilteredAllSettings Struct Reference . . . 274
0.15.38.1 Field Documentation . . . 274
0.15.39 smtrat::mcsat::onecell::BCFilteredBoundsSettings Struct Reference . . . 274
0.15.39.1 Field Documentation . . . 274
0.15.40 smtrat::mcsat::onecell::BCFilteredSamplesSettings Struct Reference . . . 275
0.15.40.1 Field Documentation . . . 275
0.15.41 smtrat::mcsat::onecell::BCFilteredSettings Struct Reference . . . 275
0.15.41.1 Field Documentation . . . 276
0.15.42 smtrat::mcsat::onecell::BCSettings Struct Reference . . . 276
0.15.42.1 Field Documentation . . . 276
0.15.43 smtrat::BEModule< Settings > Class Template Reference . . . 276
0.15.43.1 Member Typedef Documentation . . . 281
0.15.43.2 Member Enumeration Documentation . . . 281
0.15.43.3 Constructor & Destructor Documentation . . . 282
0.15.43.4 Member Function Documentation . . . 282
0.15.43.5 Field Documentation . . . 300
0.15.44 benchmax::BenchmarkResult Struct Reference . . . 302
0.15.44.1 Detailed Description . . . 303
0.15.44.2 Member Function Documentation . . . 303
0.15.44.3 Field Documentation . . . 303
0.15.45 benchmax::BenchmarkSet Class Reference . . . 304
0.15.45.1 Detailed Description . . . 304
0.15.45.2 Member Function Documentation . . . 304
0.15.46 benchmax::settings::BenchmarkSettings Struct Reference . . . 305
0.15.46.1 Detailed Description . . . 305
0.15.46.2 Field Documentation . . . 305
0.15.47 smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName > Class Template Reference . . . 306
0.15.47.1 Detailed Description . . . 306
0.15.47.2 Member Typedef Documentation . . . 306
0.15.47.3 Member Function Documentation . . . 307
0.15.48 smtrat::expression::BinaryExpression Struct Reference . . . 308
0.15.48.1 Constructor & Destructor Documentation . . . 308

0.15.48.2 Field Documentation . . . 308
0.15.49 smtrat::parser::BinaryParser Struct Reference . . . 309
0.15.49.1 Detailed Description . . . 309
0.15.49.2 Member Typedef Documentation . . . 309
0.15.49.3 Constructor & Destructor Documentation . . . 309
0.15.49.4 Member Function Documentation . . . 309
0.15.49.5 Field Documentation . . . 309
0.15.50 smtrat::parser::BitvectorTheory Struct Reference . . . 310
0.15.50.1 Detailed Description . . . 310
0.15.50.2 Member Typedef Documentation . . . 310
0.15.50.3 Constructor & Destructor Documentation . . . 311
0.15.50.4 Member Function Documentation . . . 311
0.15.50.5 Field Documentation . . . 312
0.15.51 smtrat::parser::types::BitvectorTheory Struct Reference . . . 312
0.15.51.1 Detailed Description . . . 313
0.15.51.2 Member Typedef Documentation . . . 313
0.15.52 smtrat::BlastedConstr Class Reference . . . 313
0.15.52.1 Constructor & Destructor Documentation . . . 313
0.15.52.2 Member Function Documentation . . . 314
0.15.52.3 Friends And Related Function Documentation . . . 314
0.15.53 smtrat::BlastedPoly Class Reference . . . 314
0.15.53.1 Constructor & Destructor Documentation . . . 314
0.15.53.2 Member Function Documentation . . . 315
0.15.53.3 Friends And Related Function Documentation . . . 315
0.15.54 smtrat::mcsat::Bookkeeping Class Reference . . . 315
0.15.54.1 Detailed Description . . . 316
0.15.54.2 Member Function Documentation . . . 316
0.15.55 smtrat::covering_ng::formula::node_ds::BOOL Struct Reference . . . 317
0.15.55.1 Field Documentation . . . 317
0.15.56 smtrat::parser::BooleanEncodingTheory Struct Reference . . . 317
0.15.56.1 Detailed Description . . . 318
0.15.56.2 Constructor & Destructor Documentation . . . 318
0.15.56.3 Member Function Documentation . . . 318
0.15.56.4 Field Documentation . . . 319
0.15.57 Minisat::BoolOption Class Reference . . . 320
0.15.57.1 Constructor & Destructor Documentation . . . 320
0.15.57.2 Member Function Documentation . . . 320
0.15.57.3 Field Documentation . . . 321
0.15.58 smtrat::cadcells::datastructures::Bound Class Reference . . . 321
0.15.58.1 Detailed Description . . . 321
0.15.58.2 Member Function Documentation . . . 321
0.15.59 smtrat::lra::Bound< T1, T2 > Class Template Reference . . . 322

0.15.59.1 Detailed Description . . . 323
0.15.59.2 Member Typedef Documentation . . . 323
0.15.59.3 Member Enumeration Documentation . . . 324
0.15.59.4 Constructor & Destructor Documentation . . . 324
0.15.59.5 Member Function Documentation . . . 324
0.15.59.6 Friends And Related Function Documentation . . . 331
0.15.60 smtrat::mcsat::fm::Bound Struct Reference . . . 331
0.15.60.1 Constructor & Destructor Documentation . . . 332
0.15.60.2 Friends And Related Function Documentation . . . 332
0.15.60.3 Field Documentation . . . 332
0.15.61 smtrat::vb::Bound< T > Class Template Reference . . . 332
0.15.61.1 Detailed Description . . . 333
0.15.61.2 Member Enumeration Documentation . . . 333
0.15.61.3 Constructor & Destructor Documentation . . . 334
0.15.61.4 Member Function Documentation . . . 334
0.15.61.5 Friends And Related Function Documentation . . . 337
0.15.62 smtrat::Branching Struct Reference . . . 337
0.15.62.1 Detailed Description . . . 337
0.15.62.2 Constructor & Destructor Documentation . . . 337
0.15.62.3 Field Documentation . . . 338
0.15.63 smtrat::BVAnnotation Class Reference . . . 338
0.15.63.1 Constructor & Destructor Documentation . . . 339
0.15.63.2 Member Function Documentation . . . 339
0.15.63.3 Friends And Related Function Documentation . . . 340
0.15.64 smtrat::BVDirectEncoder Class Reference . . . 340
0.15.64.1 Constructor & Destructor Documentation . . . 340
0.15.64.2 Member Function Documentation . . . 340
0.15.65 smtrat::BVModule< Settings > Class Template Reference . . . 340
0.15.65.1 Member Typedef Documentation . . . 346
0.15.65.2 Member Enumeration Documentation . . . 346
0.15.65.3 Constructor & Destructor Documentation . . . 346
0.15.65.4 Member Function Documentation . . . 346
0.15.65.5 Field Documentation . . . 366
0.15.66 smtrat::cad::CAD< Settings > Class Template Reference . . . 368
0.15.66.1 Member Typedef Documentation . . . 369
0.15.66.2 Constructor & Destructor Documentation . . . 369
0.15.66.3 Member Function Documentation . . . 369
0.15.66.4 Friends And Related Function Documentation . . . 371
0.15.66.5 Field Documentation . . . 371
0.15.67 smtrat::qe::cad::CAD< Settings > Class Template Reference . . . 371
0.15.67.1 Member Typedef Documentation . . . 372
0.15.67.2 Constructor & Destructor Documentation . . . 372

0.15.67.3 Member Function Documentation . . . 372
0.15.68 smtrat::cad::CADConstraints< BT > Class Template Reference . . . 373
0.15.68.1 Member Typedef Documentation . . . 374
0.15.68.2 Constructor & Destructor Documentation . . . 374
0.15.68.3 Member Function Documentation . . . 375
0.15.68.4 Friends And Related Function Documentation . . . 376
0.15.68.5 Field Documentation . . . 376
0.15.69 smtrat::cad::CADCore< CH > Struct Template Reference . . . 377
0.15.70 smtrat::cad::CADCore< CoreHeuristic::BySample > Struct Reference . . . 377
0.15.70.1 Member Function Documentation . . . 377
0.15.71 smtrat::cad::CADCore< CoreHeuristic::EnumerateAll > Struct Reference . . . 378
0.15.71.1 Member Function Documentation . . . 378
0.15.72 smtrat::cad::CADCore< CoreHeuristic::Interleave > Struct Reference . . . 378
0.15.72.1 Member Function Documentation . . . 378
0.15.73 smtrat::cad::CADCore< CoreHeuristic::PreferProjection > Struct Reference . . . 379
0.15.73.1 Member Function Documentation . . . 379
0.15.74 smtrat::cad::CADCore< CoreHeuristic::PreferSampling > Struct Reference . . . 379
0.15.74.1 Member Function Documentation . . . 379
0.15.75 smtrat::qe::cad::CADElimination Class Reference . . . 379
0.15.75.1 Constructor & Destructor Documentation . . . 379
0.15.75.2 Member Function Documentation . . . 380
0.15.76 smtrat::cad::CADPreprocessor Class Reference . . . 380
0.15.76.1 Constructor & Destructor Documentation . . . 380
0.15.76.2 Member Function Documentation . . . 380
0.15.76.3 Friends And Related Function Documentation . . . 381
0.15.77 smtrat::cad::CADPreprocessorSettings Struct Reference . . . 381
0.15.77.1 Member Function Documentation . . . 381
0.15.77.2 Field Documentation . . . 382
0.15.78 smtrat::qe::cad::CADSettings Struct Reference . . . 382
0.15.78.1 Field Documentation . . . 382
0.15.79 smtrat::CardinalityEncoder Class Reference . . . 383
0.15.79.1 Constructor & Destructor Documentation . . . 383
0.15.79.2 Member Function Documentation . . . 383
0.15.79.3 Field Documentation . . . 384
0.15.80 smtrat::cadcells::representation::cell< H > Struct Template Reference . . . 384
0.15.80.1 Detailed Description . . . 384
0.15.80.2 Member Function Documentation . . . 384
0.15.81 smtrat::cadcells::representation::cell< CellHeuristic::BIGGEST_CELL > Struct Reference . 385
0.15.81.1 Member Function Documentation . . . 385
0.15.82 smtrat::cadcells::representation::cell< CellHeuristic::BIGGEST_CELL_APPROXIMATION > Struct Reference . . . 385
0.15.82.1 Member Function Documentation . . . 385

0.15.83 smtrat::cadcells::representation::cell< CellHeuristic::BIGGEST_CELL_EW > Struct Reference . . . 385
0.15.83.1 Member Function Documentation . . . 386
0.15.84 smtrat::cadcells::representation::cell< CellHeuristic::CHAIN_EQ > Struct Reference . . . 386
0.15.84.1 Member Function Documentation . . . 386
0.15.85 smtrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS > Struct Reference . . . 386
0.15.85.1 Member Function Documentation . . . 386
0.15.86 smtrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS_EQ > Struct Reference . . . 386
0.15.86.1 Member Function Documentation . . . 387
0.15.87 smtrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS_EW > Struct Reference . . . 387
0.15.87.1 Member Function Documentation . . . 387
0.15.88 smtrat::cadcells::operators::properties::cell_connected Struct Reference . . . 387
0.15.88.1 Member Function Documentation . . . 387
0.15.88.2 Field Documentation . . . 388
0.15.89 smtrat::cadcells::representation::approximation::CellApproximator Class Reference . . . 388
0.15.89.1 Constructor & Destructor Documentation . . . 388
0.15.89.2 Member Function Documentation . . . 388
0.15.90 smtrat::cadcells::datastructures::CellRepresentation< P > Struct Template Reference . . . 388
0.15.90.1 Detailed Description . . . 389
0.15.90.2 Constructor & Destructor Documentation . . . 389
0.15.90.3 Field Documentation . . . 389
0.15.91 benchmax::slurm::ChunkedSubmitfileProperties Struct Reference . . . 389
0.15.91.1 Detailed Description . . . 390
0.15.91.2 Field Documentation . . . 390
0.15.92 Minisat::Clause Class Reference . . . 391
0.15.92.1 Member Function Documentation . . . 391
0.15.92.2 Friends And Related Function Documentation . . . 393
0.15.92.3 Field Documentation . . . 393
0.15.93 Minisat::ClauseAllocator Class Reference . . . 393
0.15.93.1 Member Typedef Documentation . . . 394
0.15.93.2 Member Enumeration Documentation . . . 394
0.15.93.3 Constructor & Destructor Documentation . . . 394
0.15.93.4 Member Function Documentation . . . 395
0.15.93.5 Field Documentation . . . 396
0.15.94 smtrat::mcsat::ClauseChain Class Reference . . . 396
0.15.94.1 Detailed Description . . . 397
0.15.94.2 Member Typedef Documentation . . . 397
0.15.94.3 Constructor & Destructor Documentation . . . 397
0.15.94.4 Member Function Documentation . . . 397
0.15.94.5 Friends And Related Function Documentation . . . 398

0.15.95 smtrat::sat::detail::ClauseChecker Struct Reference . . . 398
0.15.95.1 Member Function Documentation . . . 398
0.15.96 smtrat::CMakeOptionPrinter Struct Reference . . . 399
0.15.96.1 Field Documentation . . . 399
0.15.97 Minisat::CMap< T > Class Template Reference . . . 399
0.15.97.1 Member Function Documentation . . . 399
0.15.98 smtrat::CNFerModule Class Reference . . . 400
0.15.98.1 Member Enumeration Documentation . . . 406
0.15.98.2 Constructor & Destructor Documentation . . . 407
0.15.98.3 Member Function Documentation . . . 407
0.15.98.4 Field Documentation . . . 425
0.15.99 smtrat::CoCoAGBModule< Settings > Class Template Reference . . . 427
0.15.99.1 Member Enumeration Documentation . . . 432
0.15.99.2 Member Function Documentation . . . 433
0.15.99.3 Field Documentation . . . 451
0.15.100 smtrat::CollectionWithOrigins< Element, Origin > Class Template Reference . . . 454
0.15.100.1 Member Typedef Documentation . . . 454
0.15.100.2 Constructor & Destructor Documentation . . . 454
0.15.100.3 Member Function Documentation . . . 454
0.15.101 smtrat::ICPModule< Settings >::comp Struct Reference . . . 455
0.15.101.1 Detailed Description . . . 455
0.15.101.2 Member Function Documentation . . . 455
0.15.102 smtrat::cadcells::datastructures::CompoundMaxMin Struct Reference . . . 456
0.15.102.1 Detailed Description . . . 456
0.15.102.2 Member Function Documentation . . . 456
0.15.102.3 Field Documentation . . . 456
0.15.103 smtrat::cadcells::datastructures::CompoundMinMax Struct Reference . . . 456
0.15.103.1 Detailed Description . . . 456
0.15.103.2 Member Function Documentation . . . 456
0.15.103.3 Field Documentation . . . 457
0.15.104 smtrat::vs::Condition Class Reference . . . 457
0.15.104.1 Constructor & Destructor Documentation . . . 457
0.15.104.2 Member Function Documentation . . . 458
0.15.104.3 Friends And Related Function Documentation . . . 459
0.15.105 benchmax::CondorBackend Class Reference . . . 459
0.15.105.1 Detailed Description . . . 460
0.15.105.2 Member Function Documentation . . . 460
0.15.105.3 Field Documentation . . . 461
0.15.106 smtrat::mcsat::fm::ConflictGenerator< Comparator > Struct Template Reference . . . 462
0.15.106.1 Constructor & Destructor Documentation . . . 462
0.15.106.2 Member Function Documentation . . . 462
0.15.107 smtrat::cad::ConflictGraph Class Reference . . . 462

0.15.107.1 Detailed Description . . . 463
0.15.107.2 Constructor & Destructor Documentation . . . 463
0.15.107.3 Member Function Documentation . . . 463
0.15.107.4 Friends And Related Function Documentation . . . 464
0.15.108 smtrat::parser::Theories::ConstantAdder Struct Reference . . . 464
0.15.108.1 Detailed Description . . . 464
0.15.108.2 Constructor & Destructor Documentation . . . 464
0.15.108.3 Member Function Documentation . . . 464
0.15.108.4 Field Documentation . . . 465
0.15.109 smtrat::covering_ng::formula::node_ds::CONSTRAINT Struct Reference . . . 465
0.15.109.1 Field Documentation . . . 465
0.15.110 smtrat::cad::CADConstraints< BT >::ConstraintComparator Struct Reference . . . 465
0.15.110.1 Member Function Documentation . . . 465
0.15.111 smtrat::cad::preprocessor::ConstraintUpdate Struct Reference . . . 465
0.15.111.1 Field Documentation . . . 466
0.15.112 smtrat::ConstrTree Class Reference . . . 466
0.15.112.1 Constructor & Destructor Documentation . . . 466
0.15.112.2 Member Function Documentation . . . 466
0.15.113 smtrat::LRAModule< Settings >::Context Struct Reference . . . 466
0.15.113.1 Detailed Description . . . 467
0.15.113.2 Constructor & Destructor Documentation . . . 467
0.15.113.3 Field Documentation . . . 467
0.15.114 smtrat::icp::ContractionCandidate Class Reference . . . 467
0.15.114.1 Constructor & Destructor Documentation . . . 468
0.15.114.2 Member Function Documentation . . . 469
0.15.115 smtrat::icp::contractionCandidateComp Struct Reference . . . 472
0.15.115.1 Member Function Documentation . . . 472
0.15.116 smtrat::icp::ContractionCandidateManager Class Reference . . . 472
0.15.116.1 Constructor & Destructor Documentation . . . 472
0.15.116.2 Member Function Documentation . . . 472
0.15.117 smtrat::parser::conversion::Converter< To > Struct Template Reference . . . 474
0.15.117.1 Member Function Documentation . . . 474
0.15.118 smtrat::parser::conversion::Converter< FormulaT > Struct Reference . . . 474
0.15.118.1 Member Function Documentation . . . 474
0.15.119 smtrat::parser::conversion::Converter< Poly > Struct Reference . . . 475
0.15.119.1 Member Function Documentation . . . 475
0.15.120 smtrat::parser::conversion::Converter< types::BVTerm > Struct Reference . . . 475
0.15.120.1 Member Function Documentation . . . 476
0.15.121 benchmax::settings::CoreSettings Struct Reference . . . 476
0.15.121.1 Detailed Description . . . 477
0.15.121.2 Field Documentation . . . 477
0.15.122 smtrat::settings::CoreSettings Struct Reference . . . 477

0.15.122.1 Field Documentation . . . 477
0.15.123 smtrat::parser::CoreTheory Struct Reference . . . 478
0.15.123.1 Detailed Description . . . 479
0.15.123.2 Member Typedef Documentation . . . 479
0.15.123.3 Constructor & Destructor Documentation . . . 479
0.15.123.4 Member Function Documentation . . . 479
0.15.123.5 Field Documentation . . . 480
0.15.124 smtrat::parser::types::CoreTheory Struct Reference . . . 480
0.15.124.1 Detailed Description . . . 481
0.15.124.2 Member Typedef Documentation . . . 481
0.15.125 smtrat::cadcells::representation::covering< H > Struct Template Reference . . . 481
0.15.125.1 Member Function Documentation . . . 481
0.15.126 smtrat::mcsat::arithmetic::Covering Class Reference . . . 481
0.15.126.1 Detailed Description . . . 482
0.15.126.2 Constructor & Destructor Documentation . . . 482
0.15.126.3 Member Function Documentation . . . 482
0.15.126.4 Friends And Related Function Documentation . . . 482
0.15.127 smtrat::cadcells::representation::covering< CoveringHeuristic::BIGGEST_CELL_↩ COVERING > Struct Reference . . . 483
0.15.127.1 Member Function Documentation . . . 483
0.15.128 smtrat::cadcells::representation::covering< CoveringHeuristic::BIGGEST_CELL_↩ COVERING_EW > Struct Reference . . . 483
0.15.128.1 Member Function Documentation . . . 483
0.15.129 smtrat::cadcells::representation::covering< CoveringHeuristic::BIGGEST_CELL_↩ COVERING_MIN_TDEG > Struct Reference . . . 483
0.15.129.1 Member Function Documentation . . . 483
0.15.130 smtrat::cadcells::representation::covering< CoveringHeuristic::CHAIN_COVERING > Struct Reference . . . 484
0.15.130.1 Member Function Documentation . . . 484
0.15.131 smtrat::cadcells::datastructures::CoveringDescription Class Reference . . . 484
0.15.131.1 Detailed Description . . . 484
0.15.131.2 Member Function Documentation . . . 484
0.15.132 smtrat::CoveringNGModule< Settings > Class Template Reference . . . 485
0.15.132.1 Member Typedef Documentation . . . 490
0.15.132.2 Member Enumeration Documentation . . . 490
0.15.132.3 Constructor & Destructor Documentation . . . 490
0.15.132.4 Member Function Documentation . . . 490
0.15.132.5 Field Documentation . . . 509
0.15.133 smtrat::cadcells::datastructures::CoveringRepresentation< P > Struct Template Reference 511
0.15.133.1 Detailed Description . . . 511
0.15.133.2 Member Function Documentation . . . 511
0.15.133.3 Field Documentation . . . 512
0.15.134 smtrat::covering_ng::CoveringResult< op > Struct Template Reference . . . 512

0.15.134.1 Member Enumeration Documentation . . . 512
0.15.134.2 Constructor & Destructor Documentation . . . 513
0.15.134.3 Member Function Documentation . . . 513
0.15.134.4 Field Documentation . . . 514
0.15.135 smtrat::mcsat::onecellcad::recursive::CoverNullification Struct Reference . . . 514
0.15.135.1 Field Documentation . . . 514
0.15.136 smtrat::CSplitModule< Settings > Class Template Reference . . . 514
0.15.136.1 Member Typedef Documentation . . . 519
0.15.136.2 Member Enumeration Documentation . . . 519
0.15.136.3 Constructor & Destructor Documentation . . . 519
0.15.136.4 Member Function Documentation . . . 520
0.15.136.5 Field Documentation . . . 538
0.15.137 smtrat::CubeLIAModule< Settings > Class Template Reference . . . 540
0.15.137.1 Member Typedef Documentation . . . 545
0.15.137.2 Member Enumeration Documentation . . . 546
0.15.137.3 Constructor & Destructor Documentation . . . 546
0.15.137.4 Member Function Documentation . . . 546
0.15.137.5 Field Documentation . . . 564
0.15.138 smtrat::CurryModule< Settings > Class Template Reference . . . 567
0.15.138.1 Member Typedef Documentation . . . 572
0.15.138.2 Member Enumeration Documentation . . . 572
0.15.138.3 Constructor & Destructor Documentation . . . 572
0.15.138.4 Member Function Documentation . . . 572
0.15.138.5 Field Documentation . . . 593
0.15.139 smtrat::CycleEnumerator< FHG, Collector > Struct Template Reference . . . 595
0.15.139.1 Detailed Description . . . 596
0.15.139.2 Constructor & Destructor Documentation . . . 596
0.15.139.3 Member Function Documentation . . . 596
0.15.140 benchmax::Database Class Reference . . . 596
0.15.140.1 Detailed Description . . . 597
0.15.140.2 Member Typedef Documentation . . . 597
0.15.140.3 Member Function Documentation . . . 597
0.15.141 benchmax::DBAL Class Reference . . . 598
0.15.141.1 Member Typedef Documentation . . . 599
0.15.141.2 Member Function Documentation . . . 599
0.15.142 smtrat::parser::DecimalParser Struct Reference . . . 601
0.15.142.1 Detailed Description . . . 601
0.15.143 Minisat::DeepEqual< K > Struct Template Reference . . . 601
0.15.143.1 Member Function Documentation . . . 602
0.15.144 Minisat::DeepHash< K > Struct Template Reference . . . 602
0.15.144.1 Member Function Documentation . . . 602
0.15.145 smtrat::mcsat::fm::DefaultComparator Struct Reference . . . 602

0.15.145.1 Detailed Description . . . 602
0.15.145.2 Member Function Documentation . . . 602
0.15.145.3 Field Documentation . . . 602
0.15.146 smtrat::mcsat::fm::DefaultSettings Struct Reference . . . 603
0.15.146.1 Field Documentation . . . 603
0.15.147 smtrat::mcsat::smtaf::DefaultSettings Struct Reference . . . 603
0.15.147.1 Field Documentation . . . 603
0.15.148 smtrat::mcsat::vs::DefaultSettings Struct Reference . . . 603
0.15.148.1 Field Documentation . . . 603
0.15.149 smtrat::cad::projection_compare::degree Struct Reference . . . 604
0.15.150 smtrat::mcsat::onecellcad::recursive::DegreeAscending Struct Reference . . . 604
0.15.150.1 Field Documentation . . . 604
0.15.151 smtrat::analyzer::DegreeCollector Struct Reference . . . 604
0.15.151.1 Member Function Documentation . . . 604
0.15.151.2 Field Documentation . . . 604
0.15.152 smtrat::mcsat::onecellcad::recursive::DegreeDescending Struct Reference . . . 605
0.15.152.1 Field Documentation . . . 605
0.15.153 smtrat::cadcells::datastructures::DelineatedDerivation< Properties > Class Template Reference . . . 605
0.15.153.1 Detailed Description . . . 605
0.15.153.2 Constructor & Destructor Documentation . . . 606
0.15.153.3 Member Function Documentation . . . 606
0.15.154 smtrat::cadcells::datastructures::Delineation Class Reference . . . 607
0.15.154.1 Detailed Description . . . 608
0.15.154.2 Constructor & Destructor Documentation . . . 608
0.15.154.3 Member Function Documentation . . . 608
0.15.154.4 Friends And Related Function Documentation . . . 609
0.15.155 smtrat::cadcells::datastructures::DelineationInterval Class Reference . . . 609
0.15.155.1 Detailed Description . . . 609
0.15.155.2 Member Function Documentation . . . 609
0.15.155.3 Friends And Related Function Documentation . . . 610
0.15.156 smtrat::mcsat::icp::Dependencies Class Reference . . . 610
0.15.156.1 Member Function Documentation . . . 610
0.15.157 smtrat::cadcells::datastructures::DerivationRef< Properties > Class Template Reference 611
0.15.157.1 Detailed Description . . . 611
0.15.157.2 Constructor & Destructor Documentation . . . 612
0.15.157.3 Member Function Documentation . . . 612
0.15.157.4 Friends And Related Function Documentation . . . 613
0.15.158 smtrat::mcsat::onecellcad::recursive::DontCoverNullification Struct Reference . . . 613
0.15.158.1 Field Documentation . . . 613
0.15.159 smtrat::cad::debug::DotSubgraph Struct Reference . . . 614
0.15.159.1 Constructor & Destructor Documentation . . . 614

0.15.159.2 Member Function Documentation 614
0.15.159.3 Friends And Related Function Documentation 614
0.15.159.4 Field Documentation 614
0.15.160 Minisat::DoubleOption Class Reference 614
0.15.160.1 Constructor & Destructor Documentation 615
0.15.160.2 Member Function Documentation 615
0.15.160.3 Field Documentation 616
0.15.161 Minisat::DoubleRange Struct Reference 616
0.15.161.1 Constructor & Destructor Documentation 616
0.15.161.2 Field Documentation 616
0.15.162 smtrat::expression::simplifier::DuplicateSimplifier Struct Reference 617
0.15.162.1 Member Function Documentation 617
0.15.163 smtrat::DynamicPriorityQueue< T, Compare > Class Template Reference 618
0.15.163.1 Member Typedef Documentation 619
0.15.163.2 Constructor & Destructor Documentation 619
0.15.163.3 Member Function Documentation 619
0.15.163.4 Field Documentation 620
0.15.164 smtrat::cad::ProjectionGlobalInformation::ECData Struct Reference 620
0.15.164.1 Field Documentation 620
0.15.165 smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Edge Struct Reference 621
0.15.165.1 Detailed Description 621
0.15.165.2 Constructor & Destructor Documentation 621
0.15.165.3 Member Function Documentation 621
0.15.165.4 Field Documentation 622
0.15.166 smtrat::ElementWithOrigins< Element, Origin > Class Template Reference 622
0.15.166.1 Constructor & Destructor Documentation 623
0.15.166.2 Member Function Documentation 623
0.15.167 smtrat::EMModule< Settings > Class Template Reference 623
0.15.167.1 Member Typedef Documentation 628
0.15.167.2 Member Enumeration Documentation 628
0.15.167.3 Constructor & Destructor Documentation 629
0.15.167.4 Member Function Documentation 629
0.15.167.5 Field Documentation 647
0.15.168 smtrat::subtropical::Encoding Class Reference 649
0.15.168.1 Member Function Documentation 650
0.15.169 smtrat::EQGraph< VertexName > Struct Template Reference 650
0.15.169.1 Member Typedef Documentation 651
0.15.169.2 Member Function Documentation 651
0.15.169.3 Field Documentation 652
0.15.170 Minisat::Equal< K > Struct Template Reference 653
0.15.170.1 Member Function Documentation 653

0.15.171 smtrat::parser::ErrorHandler Struct Reference . . . 653
0.15.171.1 Member Function Documentation . . . 653
0.15.172 smtrat::ESModule< Settings > Class Template Reference . . . 653
0.15.172.1 Member Typedef Documentation . . . 658
0.15.172.2 Member Enumeration Documentation . . . 658
0.15.172.3 Constructor & Destructor Documentation . . . 659
0.15.172.4 Member Function Documentation . . . 659
0.15.172.5 Field Documentation . . . 677
0.15.173 smtrat::ExactlyOneCommanderEncoder Class Reference . . . 679
0.15.173.1 Constructor & Destructor Documentation . . . 680
0.15.173.2 Member Function Documentation . . . 680
0.15.173.3 Field Documentation . . . 681
0.15.174 smtrat::execution::ExecutionState Class Reference . . . 681
0.15.174.1 Constructor & Destructor Documentation . . . 681
0.15.174.2 Member Function Documentation . . . 682
0.15.175 smtrat::Executor< Strategy > Class Template Reference . . . 684
0.15.175.1 Member Typedef Documentation . . . 685
0.15.175.2 Constructor & Destructor Documentation . . . 685
0.15.175.3 Member Function Documentation . . . 686
0.15.175.4 Field Documentation . . . 690
0.15.176 smtrat::covering_ng::formula::ExhaustiveImplicants Class Reference . . . 690
0.15.176.1 Member Typedef Documentation . . . 691
0.15.176.2 Constructor & Destructor Documentation . . . 691
0.15.176.3 Member Function Documentation . . . 691
0.15.177 smtrat::mcsat::fm::Explanation< Settings > Struct Template Reference . . . 692
0.15.177.1 Member Function Documentation . . . 692
0.15.178 smtrat::mcsat::icp::Explanation Struct Reference . . . 692
0.15.178.1 Member Function Documentation . . . 692
0.15.179 smtrat::mcsat::nlsat::Explanation Struct Reference . . . 692
0.15.179.1 Member Function Documentation . . . 693
0.15.180 smtrat::mcsat::onecell::Explanation Struct Reference . . . 693
0.15.180.1 Member Function Documentation . . . 693
0.15.181 smtrat::mcsat::onecellcad::levelwise::Explanation< Setting1, Setting2 > Struct Template Reference . . . 694
0.15.181.1 Member Function Documentation . . . 694
0.15.182 smtrat::mcsat::onecellcad::recursive::Explanation< Setting1, Setting2 > Struct Template Reference . . . 694
0.15.182.1 Member Function Documentation . . . 694
0.15.183 smtrat::mcsat::vs::Explanation Struct Reference . . . 694
0.15.183.1 Member Function Documentation . . . 694
0.15.184 smtrat::mcsat::nlsat::ExplanationGenerator Class Reference . . . 695
0.15.184.1 Constructor & Destructor Documentation . . . 695
0.15.184.2 Member Function Documentation . . . 695

0.15.185 smtrat::mcsat::vs::ExplanationGenerator< Settings > Class Template Reference 695
0.15.185.1 Constructor & Destructor Documentation 696
0.15.185.2 Member Function Documentation 696
0.15.186 smtrat::expression::Expression Class Reference 696
0.15.186.1 Constructor & Destructor Documentation 697
0.15.186.2 Member Function Documentation 697
0.15.186.3 Friends And Related Function Documentation 699
0.15.187 smtrat::expression::ExpressionContent Struct Reference 699
0.15.187.1 Member Typedef Documentation 699
0.15.187.2 Constructor & Destructor Documentation 699
0.15.187.3 Friends And Related Function Documentation 699
0.15.187.4 Field Documentation 700
0.15.188 smtrat::expression::ExpressionConverter Struct Reference 700
0.15.188.1 Member Function Documentation 700
0.15.188.2 Field Documentation 701
0.15.189 smtrat::expression::ExpressionModifier Class Reference 701
0.15.189.1 Member Typedef Documentation 701
0.15.189.2 Member Function Documentation 701
0.15.190 smtrat::expression::ExpressionPool Class Reference 702
0.15.190.1 Constructor & Destructor Documentation 703
0.15.190.2 Member Function Documentation 703
0.15.190.3 Friends And Related Function Documentation 704
0.15.191 smtrat::parser::ParserState::ExpressionScope Struct Reference 704
0.15.191.1 Constructor & Destructor Documentation 704
0.15.191.2 Member Function Documentation 704
0.15.192 smtrat::expression::ExpressionTypeChecker< T > Struct Template Reference 704
0.15.192.1 Member Function Documentation 705
0.15.193 smtrat::expression::ExpressionVisitor Class Reference 705
0.15.193.1 Member Typedef Documentation 705
0.15.193.2 Member Function Documentation 705
0.15.194 smtrat::covering_ng::formula::node_ds::FALSE Struct Reference 706
0.15.195 smtrat::mcsat::FastParallelExplanation< Backends > Struct Template Reference 706
0.15.195.1 Detailed Description 706
0.15.196 smtrat::mcsat::variableordering::detail::FeatureCollector< Objects > Struct Template Reference 706
0.15.196.1 Detailed Description 707
0.15.196.2 Member Typedef Documentation 707
0.15.196.3 Member Function Documentation 707
0.15.196.4 Field Documentation 707
0.15.197 smtrat::parser::FixedWidthConstant< T > Struct Template Reference 708
0.15.197.1 Detailed Description 708
0.15.197.2 Constructor & Destructor Documentation 708

0.15.197.3 Member Function Documentation . . . 708
0.15.197.4 Field Documentation . . . 708
0.15.198 smtrat::parseformula::FormulaCollector Class Reference . . . 708
0.15.198.1 Member Typedef Documentation . . . 710
0.15.198.2 Member Function Documentation . . . 710
0.15.198.3 Field Documentation . . . 714
0.15.199 smtrat::ICPModule< Settings >::formulaPtrComp Struct Reference . . . 714
0.15.199.1 Member Function Documentation . . . 714
0.15.200 smtrat::FormulaWithOrigins Class Reference . . . 715
0.15.200.1 Detailed Description . . . 715
0.15.200.2 Constructor & Destructor Documentation . . . 715
0.15.200.3 Member Function Documentation . . . 716
0.15.200.4 Friends And Related Function Documentation . . . 716
0.15.201 smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices > Class Template Reference . . . 717
0.15.201.1 Detailed Description . . . 718
0.15.201.2 Member Typedef Documentation . . . 718
0.15.201.3 Member Function Documentation . . . 718
0.15.201.4 Friends And Related Function Documentation . . . 720
0.15.202 smtrat::FouMoModule< Settings > Class Template Reference . . . 720
0.15.202.1 Detailed Description . . . 725
0.15.202.2 Member Typedef Documentation . . . 726
0.15.202.3 Member Enumeration Documentation . . . 726
0.15.202.4 Constructor & Destructor Documentation . . . 726
0.15.202.5 Member Function Documentation . . . 726
0.15.202.6 Field Documentation . . . 746
0.15.203 smtrat::qe::fm::FourierMotzkinQE Class Reference . . . 748
0.15.203.1 Detailed Description . . . 748
0.15.203.2 Member Typedef Documentation . . . 748
0.15.203.3 Constructor & Destructor Documentation . . . 748
0.15.203.4 Member Function Documentation . . . 749
0.15.204 smtrat::FPPModule< Settings > Class Template Reference . . . 749
0.15.204.1 Member Typedef Documentation . . . 754
0.15.204.2 Member Enumeration Documentation . . . 754
0.15.204.3 Constructor & Destructor Documentation . . . 754
0.15.204.4 Member Function Documentation . . . 754
0.15.204.5 Field Documentation . . . 774
0.15.205 smtrat::mcsat::FullParallelExplanation< Backends > Struct Template Reference . . . 777
0.15.205.1 Detailed Description . . . 777
0.15.206 smtrat::cad::FullSampleComparator< Iterator, Strategy > Struct Template Reference . . . 777
0.15.207 smtrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::T > Struct Template Reference . . . 777

0.15.208 smtrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::Type > Struct Template Reference . . . 777
0.15.209 smtrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::Value > Struct Template Reference . . . 777
0.15.210 smtrat::parser::FunctionInstantiator Struct Reference . . . 777
0.15.210.1 Constructor & Destructor Documentation . . . 778
0.15.210.2 Member Function Documentation . . . 778
0.15.211 smtrat::GBModule< Settings > Class Template Reference . . . 778
0.15.211.1 Detailed Description . . . 784
0.15.211.2 Member Typedef Documentation . . . 784
0.15.211.3 Member Enumeration Documentation . . . 785
0.15.211.4 Constructor & Destructor Documentation . . . 785
0.15.211.5 Member Function Documentation . . . 785
0.15.211.6 Friends And Related Function Documentation . . . 805
0.15.211.7 Field Documentation . . . 806
0.15.212 smtrat::GBModuleState< Settings > Class Template Reference . . . 809
0.15.212.1 Detailed Description . . . 809
0.15.212.2 Constructor & Destructor Documentation . . . 809
0.15.212.3 Member Function Documentation . . . 809
0.15.212.4 Field Documentation . . . 810
0.15.213 smtrat::GBPPModule< Settings > Class Template Reference . . . 810
0.15.213.1 Member Typedef Documentation . . . 815
0.15.213.2 Member Enumeration Documentation . . . 815
0.15.213.3 Constructor & Destructor Documentation . . . 815
0.15.213.4 Member Function Documentation . . . 815
0.15.213.5 Field Documentation . . . 836
0.15.214 Minisat::Hash< K > Struct Template Reference . . . 838
0.15.214.1 Member Function Documentation . . . 838
0.15.215 std::hash< const smtrat::expression::ExpressionContent * > Struct Reference . . . 839
0.15.215.1 Member Function Documentation . . . 839
0.15.216 std::hash< smtrat::expression::BinaryExpression > Struct Reference . . . 839
0.15.216.1 Member Function Documentation . . . 839
0.15.217 std::hash< smtrat::expression::Expression > Struct Reference . . . 839
0.15.217.1 Member Function Documentation . . . 839
0.15.218 std::hash< smtrat::expression::ITEExpression > Struct Reference . . . 839
0.15.218.1 Member Function Documentation . . . 839
0.15.219 std::hash< smtrat::expression::NaryExpression > Struct Reference . . . 840
0.15.219.1 Member Function Documentation . . . 840
0.15.220 std::hash< smtrat::expression::QuantifierExpression > Struct Reference . . . 840
0.15.220.1 Member Function Documentation . . . 840
0.15.221 std::hash< smtrat::expression::UnaryExpression > Struct Reference . . . 840
0.15.221.1 Member Function Documentation . . . 840
0.15.222 std::hash< smtrat::lra::Variable< T1, T2 > > Struct Template Reference . . . 840

0.15.222.1 Detailed Description . . . 841
0.15.222.2 Member Function Documentation . . . 841
0.15.223 std::hash< smtrat::vs::Substitution > Struct Reference . . . 841
0.15.223.1 Member Function Documentation . . . 841
0.15.224 std::hash< std::vector< T > > Struct Template Reference . . . 841
0.15.224.1 Member Function Documentation . . . 841
0.15.225 std::hash_combiner Struct Reference . . . 842
0.15.225.1 Constructor & Destructor Documentation . . . 842
0.15.225.2 Member Function Documentation . . . 842
0.15.225.3 Field Documentation . . . 842
0.15.226 Minisat::Heap< Comp > Class Template Reference . . . 842
0.15.226.1 Constructor & Destructor Documentation . . . 843
0.15.226.2 Member Function Documentation . . . 843
0.15.227 smtrat::parser::HexadecimalParser Struct Reference . . . 844
0.15.227.1 Detailed Description . . . 844
0.15.227.2 Member Typedef Documentation . . . 844
0.15.227.3 Constructor & Destructor Documentation . . . 844
0.15.227.4 Member Function Documentation . . . 844
0.15.227.5 Field Documentation . . . 845
0.15.228 smtrat::icp::HistoryNode Class Reference . . . 845
0.15.228.1 Constructor & Destructor Documentation . . . 846
0.15.228.2 Member Function Documentation . . . 846
0.15.229 smtrat::ICEModule< Settings > Class Template Reference . . . 849
0.15.229.1 Member Typedef Documentation . . . 854
0.15.229.2 Member Enumeration Documentation . . . 854
0.15.229.3 Constructor & Destructor Documentation . . . 854
0.15.229.4 Member Function Documentation . . . 855
0.15.229.5 Field Documentation . . . 874
0.15.230 smtrat::ICPModule< Settings > Class Template Reference . . . 876
0.15.230.1 Member Typedef Documentation . . . 881
0.15.230.2 Member Enumeration Documentation . . . 881
0.15.230.3 Constructor & Destructor Documentation . . . 881
0.15.230.4 Member Function Documentation . . . 882
0.15.230.5 Field Documentation . . . 900
0.15.231 smtrat::icp::IcpVariable Class Reference . . . 903
0.15.231.1 Constructor & Destructor Documentation . . . 903
0.15.231.2 Member Function Documentation . . . 904
0.15.231.3 Friends And Related Function Documentation . . . 906
0.15.232 smtrat::icp::icpVariableComp Struct Reference . . . 906
0.15.232.1 Member Function Documentation . . . 906
0.15.233 smtrat::parser::Identifier Struct Reference . . . 906
0.15.233.1 Constructor & Destructor Documentation . . . 907

0.15.233.2 Member Function Documentation . . . 907
0.15.233.3 Field Documentation . . . 907
0.15.234 smtrat::parser::IdentifierParser Struct Reference . . . 908
0.15.234.1 Constructor & Destructor Documentation . . . 908
0.15.234.2 Field Documentation . . . 908
0.15.235 smtrat::cad::debug::IDSanitizer Struct Reference . . . 908
0.15.235.1 Constructor & Destructor Documentation . . . 908
0.15.235.2 Member Function Documentation . . . 908
0.15.236 smtrat::covering_ng::formula::node_ds::IFF Struct Reference . . . 909
0.15.236.1 Field Documentation . . . 909
0.15.237 smtrat::mcsat::fm::IgnoreCoreSettings Struct Reference . . . 909
0.15.237.1 Field Documentation . . . 909
0.15.238 smtrat::IncWidthModule< Settings > Class Template Reference . . . 909
0.15.238.1 Member Typedef Documentation . . . 914
0.15.238.2 Member Enumeration Documentation . . . 914
0.15.238.3 Constructor & Destructor Documentation . . . 914
0.15.238.4 Member Function Documentation . . . 915
0.15.238.5 Field Documentation . . . 933
0.15.239 smtrat::parser::IndexedFunctionInstantiator Struct Reference . . . 935
0.15.239.1 Constructor & Destructor Documentation . . . 936
0.15.239.2 Member Function Documentation . . . 936
0.15.240 smtrat::cadcells::datastructures::IndexedRoot Struct Reference . . . 936
0.15.240.1 Detailed Description . . . 936
0.15.240.2 Constructor & Destructor Documentation . . . 936
0.15.240.3 Field Documentation . . . 937
0.15.241 smtrat::cadcells::datastructures::IndexedRootOrdering Class Reference . . . 937
0.15.241.1 Detailed Description . . . 937
0.15.241.2 Member Function Documentation . . . 937
0.15.242 smtrat::cadcells::datastructures::IndexedRootRelation Struct Reference . . . 938
0.15.242.1 Detailed Description . . . 939
0.15.242.2 Field Documentation . . . 939
0.15.243 smtrat::InequalitiesTable< Settings > Class Template Reference . . . 939
0.15.243.1 Detailed Description . . . 940
0.15.243.2 Member Typedef Documentation . . . 940
0.15.243.3 Constructor & Destructor Documentation . . . 941
0.15.243.4 Member Function Documentation . . . 941
0.15.243.5 Field Documentation . . . 942
0.15.244 smtrat::lra::Bound< T1, T2 >::Info Struct Reference . . . 942
0.15.244.1 Detailed Description . . . 943
0.15.244.2 Constructor & Destructor Documentation . . . 943
0.15.244.3 Field Documentation . . . 943
0.15.245 smtrat::mcsat::InformationGetter Struct Reference . . . 943

0.15.245.1 Field Documentation . . . 944
0.15.246 smtrat::parser::Instantiator< V, T > Struct Template Reference . . . 945
0.15.246.1 Member Function Documentation . . . 946
0.15.246.2 Field Documentation . . . 946
0.15.247 smtrat::parser::InstructionHandler Class Reference . . . 947
0.15.247.1 Member Typedef Documentation . . . 948
0.15.247.2 Constructor & Destructor Documentation . . . 948
0.15.247.3 Member Function Documentation . . . 948
0.15.247.4 Field Documentation . . . 952
0.15.248 Minisat::Int64Range Struct Reference . . . 952
0.15.248.1 Constructor & Destructor Documentation . . . 953
0.15.248.2 Field Documentation . . . 953
0.15.249 smtrat::IntBlastModule< Settings > Class Template Reference . . . 953
0.15.249.1 Member Typedef Documentation . . . 958
0.15.249.2 Member Enumeration Documentation . . . 958
0.15.249.3 Constructor & Destructor Documentation . . . 958
0.15.249.4 Member Function Documentation . . . 959
0.15.249.5 Field Documentation . . . 978
0.15.250 smtrat::IntEqModule< Settings > Class Template Reference . . . 980
0.15.250.1 Detailed Description . . . 985
0.15.250.2 Member Typedef Documentation . . . 986
0.15.250.3 Member Enumeration Documentation . . . 986
0.15.250.4 Constructor & Destructor Documentation . . . 986
0.15.250.5 Member Function Documentation . . . 986
0.15.250.6 Field Documentation . . . 1006
0.15.251 smtrat::cadcells::representation::util::IntervalCompare< T > Struct Template Reference . 1008
0.15.251.1 Member Function Documentation . . . 1008
0.15.252 smtrat::covering_ng::IntervalCompare< op > Struct Template Reference . . . 1008
0.15.252.1 Detailed Description . . . 1008
0.15.252.2 Member Function Documentation . . . 1009
0.15.253 smtrat::mcsat::icp::IntervalPropagation Class Reference . . . 1009
0.15.253.1 Constructor & Destructor Documentation . . . 1009
0.15.253.2 Member Function Documentation . . . 1009
0.15.254 Minisat::IntOption Class Reference . . . 1009
0.15.254.1 Constructor & Destructor Documentation . . . 1010
0.15.254.2 Member Function Documentation . . . 1010
0.15.254.3 Field Documentation . . . 1010
0.15.255 Minisat::IntRange Struct Reference . . . 1011
0.15.255.1 Constructor & Destructor Documentation . . . 1011
0.15.255.2 Field Documentation . . . 1011
0.15.256 smtrat::is_sample_outside< S > Struct Template Reference . . . 1011
0.15.256.1 Member Function Documentation . . . 1011

0.15.257 smtrat::is_sample_outside< IsSampleOutsideAlgorithm::DEFAULT > Struct Reference . 1012
0.15.257.1 Member Function Documentation . . . 1012
0.15.258 smtrat::is_variant< T > Struct Template Reference . . . 1012
0.15.258.1 Detailed Description . . . 1012
0.15.259 smtrat::expression::ITEExpression Struct Reference . . . 1012
0.15.259.1 Constructor & Destructor Documentation . . . 1012
0.15.259.2 Field Documentation . . . 1013
0.15.260 benchmax::RandomizationAdaptor< T >::iterator Struct Reference . . . 1013
0.15.260.1 Constructor & Destructor Documentation . . . 1013
0.15.260.2 Member Function Documentation . . . 1013
0.15.261 smtrat::ModuleInput::IteratorCompare Struct Reference . . . 1014
0.15.261.1 Member Function Documentation . . . 1014
0.15.262 benchmax::Jobs Class Reference . . . 1014
0.15.262.1 Detailed Description . . . 1014
0.15.262.2 Constructor & Destructor Documentation . . . 1014
0.15.262.3 Member Function Documentation . . . 1015
0.15.263 smtrat::parser::KeywordParser Struct Reference . . . 1015
0.15.263.1 Detailed Description . . . 1015
0.15.263.2 Constructor & Destructor Documentation . . . 1015
0.15.263.3 Field Documentation . . . 1016
0.15.264 Minisat::lbool Class Reference . . . 1016
0.15.264.1 Constructor & Destructor Documentation . . . 1016
0.15.264.2 Member Function Documentation . . . 1016
0.15.264.3 Friends And Related Function Documentation . . . 1017
0.15.265 smtrat::mcsat::onecell::LDBFilteredAllSelectiveSettings Struct Reference . . . 1017
0.15.265.1 Field Documentation . . . 1017
0.15.266 smtrat::mcsat::onecell::LDBSettings Struct Reference . . . 1017
0.15.266.1 Field Documentation . . . 1018
0.15.267 smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound Struct Reference . . . 1018
0.15.267.1 Constructor & Destructor Documentation . . . 1018
0.15.267.2 Field Documentation . . . 1019
0.15.268 smtrat::Module::Lemma Struct Reference . . . 1019
0.15.268.1 Constructor & Destructor Documentation . . . 1020
0.15.268.2 Field Documentation . . . 1020
0.15.269 Minisat::LessThan_default< T > Struct Template Reference . . . 1020
0.15.269.1 Member Function Documentation . . . 1020
0.15.270 smtrat::cad::projection_compare::level Struct Reference . . . 1020
0.15.271 smtrat::cad::sample_compare::level Struct Reference . . . 1020
0.15.272 smtrat::cad::ProjectionLevelInformation::LevelInfo Struct Reference . . . 1021
0.15.272.1 Member Function Documentation . . . 1021
0.15.272.2 Field Documentation . . . 1022
0.15.273 smtrat::mcsat::onecellcad::levelwise::LevelwiseCAD Class Reference . . . 1022

0.15.273.1 Member Function Documentation . . . 1023
0.15.273.2 Field Documentation . . . 1024
0.15.274 smtrat::LevelWiseInformation< Settings > Class Template Reference . . . 1024
0.15.274.1 Constructor & Destructor Documentation . . . 1025
0.15.274.2 Member Function Documentation . . . 1025
0.15.275 smtrat::cad::LiftingTree< Settings > Class Template Reference . . . 1027
0.15.275.1 Member Typedef Documentation . . . 1027
0.15.275.2 Constructor & Destructor Documentation . . . 1028
0.15.275.3 Member Function Documentation . . . 1028
0.15.276 smtrat::resource::Limiter Class Reference . . . 1030
0.15.276.1 Member Function Documentation . . . 1030
0.15.277 smtrat::ICPModule< Settings >::linearVariable Struct Reference . . . 1031
0.15.277.1 Field Documentation . . . 1031
0.15.278 smtrat::mcsat::ClauseChain::Link Struct Reference . . . 1031
0.15.278.1 Constructor & Destructor Documentation . . . 1031
0.15.278.2 Member Function Documentation . . . 1032
0.15.278.3 Friends And Related Function Documentation . . . 1032
0.15.279 Minisat::Lit Struct Reference . . . 1032
0.15.279.1 Member Function Documentation . . . 1032
0.15.279.2 Friends And Related Function Documentation . . . 1033
0.15.279.3 Field Documentation . . . 1033
0.15.280 benchmax::LocalBackend Class Reference . . . 1033
0.15.280.1 Detailed Description . . . 1034
0.15.280.2 Member Function Documentation . . . 1034
0.15.280.3 Field Documentation . . . 1035
0.15.281 smtrat::LOG Class Reference . . . 1035
0.15.281.1 Member Function Documentation . . . 1035
0.15.281.2 Friends And Related Function Documentation . . . 1035
0.15.281.3 Field Documentation . . . 1035
0.15.282 smtrat::parser::LogicParser Struct Reference . . . 1036
0.15.282.1 Constructor & Destructor Documentation . . . 1036
0.15.283 smtrat::LongFormulaEncoder Class Reference . . . 1036
0.15.283.1 Constructor & Destructor Documentation . . . 1036
0.15.283.2 Member Function Documentation . . . 1036
0.15.283.3 Field Documentation . . . 1037
0.15.284 smtrat::LRAModule< Settings > Class Template Reference . . . 1037
0.15.284.1 Detailed Description . . . 1043
0.15.284.2 Member Typedef Documentation . . . 1044
0.15.284.3 Member Enumeration Documentation . . . 1045
0.15.284.4 Constructor & Destructor Documentation . . . 1045
0.15.284.5 Member Function Documentation . . . 1045
0.15.284.6 Field Documentation . . . 1067

0.15.285 smtrat::LVEModule< Settings > Class Template Reference 1070
0.15.285.1 Member Typedef Documentation 1075
0.15.285.2 Member Enumeration Documentation 1075
0.15.285.3 Constructor & Destructor Documentation 1075
0.15.285.4 Member Function Documentation 1075
0.15.285.5 Field Documentation 1093
0.15.286 smtrat::Manager Class Reference 1095
0.15.286.1 Detailed Description 1097
0.15.286.2 Constructor & Destructor Documentation 1097
0.15.286.3 Member Function Documentation 1097
0.15.286.4 Friends And Related Function Documentation 1104
0.15.287 Minisat::Map< K, D, H, E > Class Template Reference 1104
0.15.287.1 Constructor & Destructor Documentation 1104
0.15.287.2 Member Function Documentation 1105
0.15.288 benchmax::MathSAT Class Reference 1106
0.15.288.1 Detailed Description 1107
0.15.288.2 Constructor & Destructor Documentation 1107
0.15.288.3 Member Function Documentation 1107
0.15.288.4 Field Documentation 1108
0.15.289 smtrat::mcsat::fm::MaxSizeComparator Struct Reference 1108
0.15.289.1 Detailed Description 1108
0.15.289.2 Member Function Documentation 1108
0.15.289.3 Field Documentation 1109
0.15.290 smtrat::MaxSMT< Solver, Strategy > Class Template Reference 1109
0.15.290.1 Constructor & Destructor Documentation 1109
0.15.290.2 Member Function Documentation 1109
0.15.291 smtrat::maxsmt::MaxSMTBackend< Solver, Strategy > Class Template Reference 1110
0.15.292 smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::FU_MALIK_INCREMENTAL > Class Template Reference 1110
0.15.292.1 Constructor & Destructor Documentation 1110
0.15.292.2 Member Function Documentation 1110
0.15.293 smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::LINEAR_SEARCH > Class Template Reference 1110
0.15.293.1 Constructor & Destructor Documentation 1110
0.15.293.2 Member Function Documentation 1110
0.15.294 smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::MSU3 > Class Template Reference 1111
0.15.294.1 Constructor & Destructor Documentation 1111
0.15.294.2 Member Function Documentation 1111
0.15.295 smtrat::MCBModule< Settings > Class Template Reference 1111
0.15.295.1 Member Typedef Documentation 1116
0.15.295.2 Member Enumeration Documentation 1116
0.15.295.3 Constructor & Destructor Documentation 1116

0.15.295.4 Member Function Documentation . . . 1117
0.15.295.5 Field Documentation . . . 1135
0.15.296 smtrat::mcsat::MCSATBackend< Settings > Class Template Reference . . . 1137
0.15.296.1 Member Function Documentation . . . 1138
0.15.297 smtrat::mcsat::MCSATMixin< Settings > Class Template Reference . . . 1139
0.15.297.1 Constructor & Destructor Documentation . . . 1140
0.15.297.2 Member Function Documentation . . . 1140
0.15.297.3 Friends And Related Function Documentation . . . 1144
0.15.298 smtrat::expression::simplifier::MergeSimplifier Struct Reference . . . 1145
0.15.298.1 Member Function Documentation . . . 1145
0.15.299 smtrat::covering_ng::formula::Minimal Class Reference . . . 1146
0.15.299.1 Member Typedef Documentation . . . 1147
0.15.299.2 Constructor & Destructor Documentation . . . 1147
0.15.299.3 Member Function Documentation . . . 1147
0.15.300 benchmax::Minisat Class Reference . . . 1147
0.15.300.1 Detailed Description . . . 1148
0.15.300.2 Constructor & Destructor Documentation . . . 1148
0.15.300.3 Member Function Documentation . . . 1148
0.15.300.4 Field Documentation . . . 1149
0.15.301 benchmax::Minisatp Class Reference . . . 1150
0.15.301.1 Detailed Description . . . 1151
0.15.301.2 Constructor & Destructor Documentation . . . 1151
0.15.301.3 Member Function Documentation . . . 1151
0.15.301.4 Field Documentation . . . 1152
0.15.302 smtrat::mcsat::fm::MinSizeComparator Struct Reference . . . 1152
0.15.302.1 Detailed Description . . . 1152
0.15.302.2 Member Function Documentation . . . 1152
0.15.302.3 Field Documentation . . . 1153
0.15.303 smtrat::mcsat::fm::MinVarCountComparator Struct Reference . . . 1153
0.15.303.1 Detailed Description . . . 1153
0.15.303.2 Member Function Documentation . . . 1153
0.15.303.3 Field Documentation . . . 1153
0.15.304 smtrat::cad::MISGeneration< heuristic > Class Template Reference . . . 1153
0.15.304.1 Member Function Documentation . . . 1153
0.15.305 smtrat::MixedSignEncoder Class Reference . . . 1154
0.15.305.1 Constructor & Destructor Documentation . . . 1155
0.15.305.2 Member Function Documentation . . . 1155
0.15.305.3 Field Documentation . . . 1155
0.15.306 smtrat::cad::ModelBasedProjection< incrementality, backtracking, Settings > Class Template Reference . . . 1156
0.15.306.1 Member Typedef Documentation . . . 1157
0.15.306.2 Member Function Documentation . . . 1157

0.15.306.3 Field Documentation . . . 1160
0.15.307 smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings > Class Template Reference . . . 1161
0.15.307.1 Detailed Description . . . 1162
0.15.307.2 Constructor & Destructor Documentation . . . 1162
0.15.307.3 Member Function Documentation . . . 1163
0.15.307.4 Friends And Related Function Documentation . . . 1166
0.15.307.5 Field Documentation . . . 1167
0.15.308 smtrat::Module Class Reference . . . 1167
0.15.308.1 Detailed Description . . . 1172
0.15.308.2 Member Enumeration Documentation . . . 1172
0.15.308.3 Constructor & Destructor Documentation . . . 1172
0.15.308.4 Member Function Documentation . . . 1173
0.15.308.5 Field Documentation . . . 1191
0.15.309 smtrat::ModuleFactory< Module > Struct Template Reference . . . 1193
0.15.309.1 Constructor & Destructor Documentation . . . 1193
0.15.309.2 Member Function Documentation . . . 1193
0.15.310 smtrat::ModuleInput Class Reference . . . 1194
0.15.310.1 Detailed Description . . . 1195
0.15.310.2 Member Typedef Documentation . . . 1195
0.15.310.3 Constructor & Destructor Documentation . . . 1195
0.15.310.4 Member Function Documentation . . . 1195
0.15.310.5 Friends And Related Function Documentation . . . 1200
0.15.311 smtrat::settings::ModuleSettings Struct Reference . . . 1200
0.15.311.1 Constructor & Destructor Documentation . . . 1200
0.15.311.2 Member Function Documentation . . . 1200
0.15.311.3 Field Documentation . . . 1201
0.15.312 smtrat::Module::ModuleStatistics Struct Reference . . . 1201
0.15.312.1 Member Function Documentation . . . 1201
0.15.313 smtrat::subtropical::Moment Struct Reference . . . 1202
0.15.313.1 Detailed Description . . . 1202
0.15.313.2 Constructor & Destructor Documentation . . . 1202
0.15.313.3 Field Documentation . . . 1202
0.15.314 smtrat::MonomialMappingByVariablePool Class Reference . . . 1202
0.15.314.1 Member Function Documentation . . . 1202
0.15.314.2 Friends And Related Function Documentation . . . 1203
0.15.315 smtrat::expression::NaryExpression Struct Reference . . . 1203
0.15.315.1 Constructor & Destructor Documentation . . . 1203
0.15.315.2 Member Function Documentation . . . 1204
0.15.315.3 Field Documentation . . . 1204
0.15.316 smtrat::expression::simplifier::NegationSimplifier Struct Reference . . . 1204
0.15.316.1 Member Function Documentation . . . 1204

0.15.317 smtrat::NewCADModule< Settings > Class Template Reference 1205
0.15.317.1 Member Typedef Documentation 1211
0.15.317.2 Member Enumeration Documentation 1211
0.15.317.3 Constructor & Destructor Documentation 1211
0.15.317.4 Member Function Documentation 1211
0.15.317.5 Field Documentation 1231
0.15.318 smtrat::NewCoveringModule< Settings > Class Template Reference 1233
0.15.318.1 Member Typedef Documentation 1239
0.15.318.2 Member Enumeration Documentation 1239
0.15.318.3 Constructor & Destructor Documentation 1239
0.15.318.4 Member Function Documentation 1239
0.15.318.5 Field Documentation 1261
0.15.319 benchmax::ssh::Node Struct Reference 1263
0.15.319.1 Detailed Description 1264
0.15.319.2 Field Documentation 1264
0.15.320 smtrat::covering_ng::formula::node_ds::Node Class Reference 1264
0.15.320.1 Constructor & Destructor Documentation 1264
0.15.320.2 Member Function Documentation 1264
0.15.321 smtrat::covering_ng::formula::node_ds::NodeContent Struct Reference 1265
0.15.321.1 Constructor & Destructor Documentation 1265
0.15.321.2 Field Documentation 1265
0.15.322 smtrat::mcsat::onecellcad::recursive::NoHeuristic Struct Reference 1266
0.15.322.1 Field Documentation 1266
0.15.323 smtrat::covering_ng::CoveringResult< op >::NONE Struct Reference 1266
0.15.324 smtrat::covering_ng::formula::node_ds::NOT Struct Reference 1266
0.15.324.1 Field Documentation 1266
0.15.325 smtrat::NRAILModule< Settings > Class Template Reference 1267
0.15.325.1 Member Typedef Documentation 1272
0.15.325.2 Member Enumeration Documentation 1272
0.15.325.3 Constructor & Destructor Documentation 1272
0.15.325.4 Member Function Documentation 1272
0.15.325.5 Field Documentation 1291
0.15.326 smtrat::parser::NumeralParser Struct Reference 1293
0.15.326.1 Detailed Description 1293
0.15.327 smtrat::lra::Numeric Class Reference 1294
0.15.327.1 Constructor & Destructor Documentation 1295
0.15.327.2 Member Function Documentation 1296
0.15.328 smtrat::execution::Objective Struct Reference 1300
0.15.328.1 Field Documentation 1300
0.15.329 Minisat::OccLists< Idx, Vec, Deleted > Class Template Reference 1300
0.15.329.1 Constructor & Destructor Documentation 1301
0.15.329.2 Member Function Documentation 1301

0.15.330 smtrat::mcsat::onecellcad::OneCellCAD Class Reference . . . 1301
0.15.330.1 Constructor & Destructor Documentation . . . 1302
0.15.330.2 Member Function Documentation . . . 1302
0.15.330.3 Field Documentation . . . 1303
0.15.331 benchmax::settings::OperationSettings Struct Reference . . . 1303
0.15.331.1 Detailed Description . . . 1304
0.15.331.2 Field Documentation . . . 1304
0.15.332 smtrat::Optimization< Solver > Class Template Reference . . . 1304
0.15.332.1 Constructor & Destructor Documentation . . . 1304
0.15.332.2 Member Function Documentation . . . 1304
0.15.333 Minisat::Option Class Reference . . . 1305
0.15.333.1 Constructor & Destructor Documentation . . . 1305
0.15.333.2 Member Function Documentation . . . 1306
0.15.333.3 Friends And Related Function Documentation . . . 1306
0.15.333.4 Field Documentation . . . 1306
0.15.334 Minisat::Option::OptionLt Struct Reference . . . 1307
0.15.334.1 Member Function Documentation . . . 1307
0.15.335 smtrat::covering_ng::formula::node_ds::OR Struct Reference . . . 1307
0.15.335.1 Field Documentation . . . 1307
0.15.336 smtrat::cad::Origin Class Reference . . . 1307
0.15.336.1 Detailed Description . . . 1308
0.15.336.2 Constructor & Destructor Documentation . . . 1308
0.15.336.3 Member Function Documentation . . . 1308
0.15.336.4 Friends And Related Function Documentation . . . 1310
0.15.337 smtrat::cad::preprocessor::Origins Struct Reference . . . 1310
0.15.337.1 Member Function Documentation . . . 1310
0.15.337.2 Field Documentation . . . 1310
0.15.338 Minisat::OutOfMemoryException Class Reference . . . 1311
0.15.339 smtrat::parser::OutputWrapper Class Reference . . . 1311
0.15.339.1 Constructor & Destructor Documentation . . . 1311
0.15.339.2 Member Function Documentation . . . 1311
0.15.340 Minisat::Map< K, D, H, E >::Pair Struct Reference . . . 1311
0.15.340.1 Field Documentation . . . 1311
0.15.341 smtrat::mcsat::ParallelExplanation< Backends > Struct Template Reference . . . 1312
0.15.341.1 Member Function Documentation . . . 1312
0.15.342 smtrat::parser::ParserSettings Struct Reference . . . 1312
0.15.342.1 Field Documentation . . . 1312
0.15.343 smtrat::parser::ParserState Struct Reference . . . 1312
0.15.343.1 Constructor & Destructor Documentation . . . 1313
0.15.343.2 Member Function Documentation . . . 1313
0.15.343.3 Field Documentation . . . 1315
0.15.344 smtrat::PBGaussModule< Settings > Class Template Reference . . . 1316

0.15.344.1 Member Typedef Documentation . . . 1321
0.15.344.2 Member Enumeration Documentation . . . 1321
0.15.344.3 Constructor & Destructor Documentation . . . 1322
0.15.344.4 Member Function Documentation . . . 1322
0.15.344.5 Field Documentation . . . 1342
0.15.345 smtrat::PBPPModule< Settings > Class Template Reference . . . 1345
0.15.345.1 Member Typedef Documentation . . . 1350
0.15.345.2 Member Enumeration Documentation . . . 1350
0.15.345.3 Constructor & Destructor Documentation . . . 1350
0.15.345.4 Member Function Documentation . . . 1350
0.15.345.5 Field Documentation . . . 1371
0.15.346 smtrat::PersistentUnionFind Struct Reference . . . 1373
0.15.346.1 Member Typedef Documentation . . . 1374
0.15.346.2 Member Function Documentation . . . 1374
0.15.346.3 Field Documentation . . . 1375
0.15.347 smtrat::PFEModule< Settings > Class Template Reference . . . 1375
0.15.347.1 Member Typedef Documentation . . . 1381
0.15.347.2 Member Enumeration Documentation . . . 1381
0.15.347.3 Constructor & Destructor Documentation . . . 1381
0.15.347.4 Member Function Documentation . . . 1381
0.15.347.5 Field Documentation . . . 1400
0.15.348 smtrat::PModule Class Reference . . . 1402
0.15.348.1 Member Enumeration Documentation . . . 1407
0.15.348.2 Constructor & Destructor Documentation . . . 1407
0.15.348.3 Member Function Documentation . . . 1407
0.15.348.4 Field Documentation . . . 1426
0.15.349 smtrat::cadcells::operators::properties::poly_additional_root_outside Struct Reference . . . 1428
0.15.349.1 Member Function Documentation . . . 1429
0.15.349.2 Field Documentation . . . 1429
0.15.350 smtrat::cadcells::operators::properties::poly_irreducible_semi_sgn_inv Struct Reference . . . 1429
0.15.350.1 Member Function Documentation . . . 1429
0.15.350.2 Field Documentation . . . 1429
0.15.351 smtrat::cadcells::operators::properties::poly_irreducible_sgn_inv Struct Reference . . . 1430
0.15.351.1 Member Function Documentation . . . 1430
0.15.351.2 Field Documentation . . . 1430
0.15.352 smtrat::cadcells::operators::properties::poly_ord_inv Struct Reference . . . 1430
0.15.352.1 Member Function Documentation . . . 1431
0.15.352.2 Field Documentation . . . 1431
0.15.353 smtrat::cadcells::operators::properties::poly_ord_inv_base Struct Reference . . . 1431
0.15.353.1 Member Function Documentation . . . 1431
0.15.353.2 Field Documentation . . . 1431
0.15.354 smtrat::cadcells::operators::properties::poly_pdel Struct Reference . . . 1432

0.15.354.1 Member Function Documentation . . . 1432
0.15.354.2 Field Documentation . . . 1432
0.15.355 smtrat::cadcells::operators::properties::poly_semi_sgn_inv Struct Reference . . . 1432
0.15.355.1 Member Function Documentation . . . 1433
0.15.355.2 Field Documentation . . . 1433
0.15.356 smtrat::cadcells::operators::properties::poly_sgn_inv Struct Reference . . . 1433
0.15.356.1 Member Function Documentation . . . 1433
0.15.356.2 Field Documentation . . . 1433
0.15.357 smtrat::cadcells::representation::util::PolyDelineation Struct Reference . . . 1434
0.15.357.1 Field Documentation . . . 1434
0.15.358 smtrat::cadcells::representation::util::PolyDelineations Struct Reference . . . 1434
0.15.358.1 Member Function Documentation . . . 1434
0.15.358.2 Field Documentation . . . 1434
0.15.359 smtrat::cad::ProjectionPolynomialInformation::PolyInfo Struct Reference . . . 1434
0.15.359.1 Field Documentation . . . 1435
0.15.360 smtrat::cad::PolynomialComparator< PolynomialGetter > Struct Template Reference . . . 1435
0.15.360.1 Constructor & Destructor Documentation . . . 1435
0.15.360.2 Member Function Documentation . . . 1435
0.15.361 smtrat::cad::PolynomialLiftingQueue< PolynomialGetter > Class Template Reference . . . 1435
0.15.361.1 Constructor & Destructor Documentation . . . 1436
0.15.361.2 Member Function Documentation . . . 1436
0.15.361.3 Friends And Related Function Documentation . . . 1436
0.15.362 smtrat::cadcells::datastructures::PolyPool Class Reference . . . 1437
0.15.362.1 Detailed Description . . . 1437
0.15.362.2 Constructor & Destructor Documentation . . . 1437
0.15.362.3 Member Function Documentation . . . 1437
0.15.362.4 Friends And Related Function Documentation . . . 1438
0.15.363 smtrat::cadcells::datastructures::detail::PolyProperties Struct Reference . . . 1438
0.15.363.1 Field Documentation . . . 1438
0.15.364 smtrat::cadcells::datastructures::PolyRef Struct Reference . . . 1439
0.15.364.1 Detailed Description . . . 1439
0.15.364.2 Field Documentation . . . 1439
0.15.365 smtrat::PolyTree Class Reference . . . 1439
0.15.365.1 Member Enumeration Documentation . . . 1439
0.15.365.2 Constructor & Destructor Documentation . . . 1440
0.15.365.3 Member Function Documentation . . . 1440
0.15.366 smtrat::PolyTreeContent Class Reference . . . 1440
0.15.366.1 Constructor & Destructor Documentation . . . 1440
0.15.366.2 Member Function Documentation . . . 1441
0.15.366.3 Friends And Related Function Documentation . . . 1441
0.15.366.4 Field Documentation . . . 1441
0.15.367 smtrat::PolyTreePool Class Reference . . . 1441

0.15.367.1 Constructor & Destructor Documentation . . . 1441
0.15.367.2 Member Function Documentation . . . 1442
0.15.368 smtrat::cad::Preprocessor Class Reference . . . 1442
0.15.368.1 Constructor & Destructor Documentation . . . 1442
0.15.368.2 Member Function Documentation . . . 1442
0.15.368.3 Friends And Related Function Documentation . . . 1443
0.15.369 smtrat::cad::PreprocessorSettings Struct Reference . . . 1443
0.15.369.1 Member Function Documentation . . . 1443
0.15.369.2 Field Documentation . . . 1444
0.15.370 benchmax::settings::PresetSettings Struct Reference . . . 1444
0.15.370.1 Field Documentation . . . 1444
0.15.371 smtrat::PriorityQueue< T, Compare > Class Template Reference . . . 1444
0.15.371.1 Constructor & Destructor Documentation . . . 1445
0.15.371.2 Member Function Documentation . . . 1445
0.15.371.3 Field Documentation . . . 1446
0.15.372 smtrat::cad::Projection< incrementality, backtracking, Settings > Class Template Reference . . . 1446
0.15.372.1 Member Typedef Documentation . . . 1447
0.15.372.2 Member Function Documentation . . . 1447
0.15.372.3 Field Documentation . . . 1451
0.15.373 smtrat::qe::cad::Projection< Settings > Class Template Reference . . . 1451
0.15.373.1 Constructor & Destructor Documentation . . . 1453
0.15.373.2 Member Function Documentation . . . 1453
0.15.373.3 Friends And Related Function Documentation . . . 1457
0.15.373.4 Field Documentation . . . 1457
0.15.374 smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings > Class Template Reference . . . 1457
0.15.374.1 Constructor & Destructor Documentation . . . 1458
0.15.374.2 Member Function Documentation . . . 1459
0.15.374.3 Friends And Related Function Documentation . . . 1462
0.15.374.4 Field Documentation . . . 1462
0.15.375 smtrat::cad::Projection< Incrementality::FULL, BT, Settings > Class Template Reference 1463
0.15.375.1 Constructor & Destructor Documentation . . . 1464
0.15.375.2 Member Function Documentation . . . 1464
0.15.375.3 Friends And Related Function Documentation . . . 1467
0.15.375.4 Field Documentation . . . 1468
0.15.376 smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings > Class Template Reference . . . 1468
0.15.376.1 Detailed Description . . . 1469
0.15.376.2 Constructor & Destructor Documentation . . . 1470
0.15.376.3 Member Function Documentation . . . 1470
0.15.376.4 Friends And Related Function Documentation . . . 1473
0.15.376.5 Field Documentation . . . 1474

0.15.377 smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings > Class Template Reference . . . 1474
0.15.377.1 Detailed Description . . . 1475
0.15.377.2 Constructor & Destructor Documentation . . . 1475
0.15.377.3 Member Function Documentation . . . 1476
0.15.377.4 Friends And Related Function Documentation . . . 1479
0.15.377.5 Field Documentation . . . 1479
0.15.378 smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings > Class Template Reference . . . 1480
0.15.378.1 Constructor & Destructor Documentation . . . 1481
0.15.378.2 Member Function Documentation . . . 1481
0.15.378.3 Friends And Related Function Documentation . . . 1485
0.15.378.4 Field Documentation . . . 1485
0.15.379 smtrat::cad::projection_compare::ProjectionComparator< Strategy > Struct Template Reference . . . 1485
0.15.380 smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::D > Struct Reference . . . 1485
0.15.380.1 Member Function Documentation . . . 1485
0.15.381 smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::ID > Struct Reference . . . 1486
0.15.381.1 Member Function Documentation . . . 1486
0.15.382 smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy↩::LD > Struct Reference . . . 1486
0.15.382.1 Member Function Documentation . . . 1486
0.15.383 smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy↩::PD > Struct Reference . . . 1486
0.15.383.1 Member Function Documentation . . . 1486
0.15.384 smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy↩::SD > Struct Reference . . . 1486
0.15.384.1 Member Function Documentation . . . 1487
0.15.385 smtrat::cad::projection_compare::ProjectionComparator_impl< Args > Struct Template Reference . . . 1487
0.15.385.1 Member Function Documentation . . . 1487
0.15.386 smtrat::cad::ProjectionGlobalInformation Class Reference . . . 1487
0.15.386.1 Member Typedef Documentation . . . 1488
0.15.386.2 Member Function Documentation . . . 1488
0.15.386.3 Field Documentation . . . 1488
0.15.387 smtrat::cad::ProjectionInformation Class Reference . . . 1488
0.15.387.1 Member Function Documentation . . . 1489
0.15.388 smtrat::cad::ProjectionLevelInformation Class Reference . . . 1490
0.15.388.1 Member Function Documentation . . . 1491
0.15.389 smtrat::cad::ProjectionOperator Struct Reference . . . 1491
0.15.389.1 Member Function Documentation . . . 1491
0.15.390 smtrat::cad::ProjectionPolynomialInformation Class Reference . . . 1492
0.15.390.1 Member Function Documentation . . . 1492

0.15.391 smtrat::cadcells::datastructures::Projections Class Reference . . . 1493
0.15.391.1 Detailed Description . . . 1493
0.15.391.2 Constructor & Destructor Documentation . . . 1493
0.15.391.3 Member Function Documentation . . . 1493
0.15.392 smtrat::cadcells::operators::mccallum_filtered_impl::PropertiesSet Struct Reference . . . 1495
0.15.392.1 Member Typedef Documentation . . . 1496
0.15.393 smtrat::cadcells::operators::PropertiesSet< Op > Struct Template Reference . . . 1496
0.15.394 smtrat::cadcells::operators::PropertiesSet< op::mccallum > Struct Reference . . . 1496
0.15.394.1 Member Typedef Documentation . . . 1496
0.15.395 smtrat::cadcells::datastructures::PropertiesT< Ts > Struct Template Reference . . . 1496
0.15.395.1 Detailed Description . . . 1496
0.15.396 smtrat::cadcells::datastructures::PropertiesT< T, Ts... > Struct Template Reference . . . 1497
0.15.396.1 Field Documentation . . . 1497
0.15.397 smtrat::cadcells::datastructures::PropertiesTContent< T, is_flag > Struct Template Reference . . . 1497
0.15.398 smtrat::cadcells::datastructures::PropertiesTContent< T, false > Struct Template Reference . . . 1497
0.15.398.1 Member Typedef Documentation . . . 1497
0.15.399 smtrat::cadcells::datastructures::PropertiesTContent< T, true > Struct Template Reference . . . 1497
0.15.399.1 Member Typedef Documentation . . . 1497
0.15.400 smtrat::cadcells::datastructures::property_hash< T > Struct Template Reference . . . 1497
0.15.400.1 Member Function Documentation . . . 1498
0.15.401 smtrat::PseudoBoolEncoder Class Reference . . . 1498
0.15.401.1 Detailed Description . . . 1498
0.15.401.2 Member Function Documentation . . . 1498
0.15.401.3 Field Documentation . . . 1499
0.15.402 smtrat::PseudoBoolNormalizer Class Reference . . . 1499
0.15.402.1 Member Function Documentation . . . 1499
0.15.403 smtrat::parser::QEParser Struct Reference . . . 1500
0.15.403.1 Constructor & Destructor Documentation . . . 1500
0.15.403.2 Member Function Documentation . . . 1500
0.15.403.3 Field Documentation . . . 1500
0.15.404 smtrat::parser::QualifiedIdentifierParser Struct Reference . . . 1501
0.15.404.1 Constructor & Destructor Documentation . . . 1501
0.15.404.2 Member Function Documentation . . . 1501
0.15.404.3 Field Documentation . . . 1501
0.15.405 smtrat::expression::QuantifierExpression Struct Reference . . . 1501
0.15.405.1 Constructor & Destructor Documentation . . . 1502
0.15.405.2 Field Documentation . . . 1502
0.15.406 smtrat::parser::QuantifierParser Struct Reference . . . 1502
0.15.406.1 Constructor & Destructor Documentation . . . 1502
0.15.407 Minisat::Queue< T > Class Template Reference . . . 1502
0.15.407.1 Member Typedef Documentation . . . 1503

0.15.407.2 Constructor & Destructor Documentation . . . 1503
0.15.407.3 Member Function Documentation . . . 1503
0.15.408 smtrat::mcsat::icp::QueueEntry Struct Reference . . . 1503
0.15.408.1 Field Documentation . . . 1504
0.15.409 benchmax::RandomizationAdaptor< T > Class Template Reference . . . 1504
0.15.409.1 Detailed Description . . . 1504
0.15.409.2 Constructor & Destructor Documentation . . . 1504
0.15.409.3 Member Function Documentation . . . 1504
0.15.410 smtrat::RationalCapsule Class Reference . . . 1505
0.15.410.1 Constructor & Destructor Documentation . . . 1505
0.15.410.2 Member Function Documentation . . . 1505
0.15.410.3 Field Documentation . . . 1505
0.15.411 smtrat::parser::RationalPolicies Struct Reference . . . 1506
0.15.411.1 Detailed Description . . . 1506
0.15.411.2 Member Function Documentation . . . 1506
0.15.412 smtrat::mcsat::onecellcad::RealAlgebraicPoint< Number > Class Template Reference . . . 1506
0.15.412.1 Detailed Description . . . 1507
0.15.412.2 Constructor & Destructor Documentation . . . 1507
0.15.412.3 Member Function Documentation . . . 1507
0.15.413 smtrat::mcsat::onecellcad::recursive::RecursiveCAD Class Reference . . . 1508
0.15.413.1 Member Function Documentation . . . 1509
0.15.413.2 Field Documentation . . . 1512
0.15.414 smtrat::cad::projection::Reducta< Poly > Struct Template Reference . . . 1512
0.15.414.1 Detailed Description . . . 1512
0.15.414.2 Constructor & Destructor Documentation . . . 1513
0.15.414.3 Field Documentation . . . 1513
0.15.415 Minisat::RegionAllocator< T > Class Template Reference . . . 1513
0.15.415.1 Member Typedef Documentation . . . 1513
0.15.415.2 Member Enumeration Documentation . . . 1513
0.15.415.3 Constructor & Destructor Documentation . . . 1514
0.15.415.4 Member Function Documentation . . . 1514
0.15.416 smtrat::parser::ErrorHandler::result< typename > Struct Template Reference . . . 1515
0.15.416.1 Member Typedef Documentation . . . 1515
0.15.417 smtrat::cad::preprocessor::ResultantRule Class Reference . . . 1515
0.15.417.1 Constructor & Destructor Documentation . . . 1515
0.15.417.2 Member Function Documentation . . . 1515
0.15.418 benchmax::Results Class Reference . . . 1516
0.15.418.1 Detailed Description . . . 1516
0.15.418.2 Member Function Documentation . . . 1516
0.15.419 smtrat::RNSEncoder Class Reference . . . 1517
0.15.419.1 Constructor & Destructor Documentation . . . 1517
0.15.419.2 Member Function Documentation . . . 1517

0.15.419.3 Field Documentation . . . 1518
0.15.420 smtrat::cadcells::operators::properties::root_ordering_holds Struct Reference . . . 1518
0.15.420.1 Member Function Documentation . . . 1518
0.15.420.2 Field Documentation . . . 1518
0.15.421 smtrat::cadcells::operators::properties::root_well_def Struct Reference . . . 1519
0.15.421.1 Member Function Documentation . . . 1519
0.15.421.2 Field Documentation . . . 1519
0.15.422 smtrat::cadcells::datastructures::RootFunction Class Reference . . . 1519
0.15.422.1 Constructor & Destructor Documentation . . . 1520
0.15.422.2 Member Function Documentation . . . 1520
0.15.422.3 Friends And Related Function Documentation . . . 1521
0.15.423 smtrat::mcsat::arithmetic::RootIndexer< RANT > Class Template Reference . . . 1521
0.15.423.1 Member Function Documentation . . . 1522
0.15.424 smtrat::cad::Sample Class Reference . . . 1522
0.15.424.1 Constructor & Destructor Documentation . . . 1523
0.15.424.2 Member Function Documentation . . . 1523
0.15.424.3 Friends And Related Function Documentation . . . 1524
0.15.425 smtrat::cad::sample_compare::SampleComparator< Iterator, Strategy > Struct Template Reference . . . 1524
0.15.426 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy↩ ::LS > Struct Template Reference . . . 1525
0.15.426.1 Member Function Documentation . . . 1525
0.15.427 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy↩ ::LT > Struct Template Reference . . . 1525
0.15.427.1 Member Function Documentation . . . 1525
0.15.428 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy↩ ::LTA > Struct Template Reference . . . 1525
0.15.428.1 Member Function Documentation . . . 1525
0.15.429 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy↩ ::LTS > Struct Template Reference . . . 1525
0.15.429.1 Member Function Documentation . . . 1526
0.15.430 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy↩ ::LTSA > Struct Template Reference . . . 1526
0.15.430.1 Member Function Documentation . . . 1526
0.15.431 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::S > Struct Template Reference . . . 1526
0.15.431.1 Member Function Documentation . . . 1526
0.15.432 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::T > Struct Template Reference . . . 1526
0.15.432.1 Member Function Documentation . . . 1527
0.15.433 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy↩ ::TLSA > Struct Template Reference . . . 1527
0.15.433.1 Member Function Documentation . . . 1527
0.15.434 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy↩ ::TS > Struct Template Reference . . . 1527

0.15.434.1 Member Function Documentation . . . 1527
0.15.435 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy↩ ::TSA > Struct Template Reference . . . 1527
0.15.435.1 Member Function Documentation . . . 1527
0.15.436 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy↩ ::Type > Struct Template Reference . . . 1528
0.15.436.1 Member Function Documentation . . . 1528
0.15.437 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy↩ ::Value > Struct Template Reference . . . 1528
0.15.437.1 Member Function Documentation . . . 1528
0.15.438 smtrat::cad::sample_compare::SampleComparator_impl< It, Args > Struct Template Reference . . . 1529
0.15.438.1 Member Function Documentation . . . 1529
0.15.439 smtrat::cadcells::datastructures::SampledDerivation< Properties > Class Template Reference . . . 1529
0.15.439.1 Detailed Description . . . 1529
0.15.439.2 Constructor & Destructor Documentation . . . 1530
0.15.439.3 Member Function Documentation . . . 1530
0.15.440 smtrat::SampledDerivationRefCompare Struct Reference . . . 1532
0.15.440.1 Member Function Documentation . . . 1532
0.15.441 smtrat::cad::SampleIteratorQueue< Iterator, Comparator > Class Template Reference . . . 1532
0.15.441.1 Member Function Documentation . . . 1532
0.15.442 smtrat::covering_ng::sampling< S > Struct Template Reference . . . 1533
0.15.442.1 Member Function Documentation . . . 1534
0.15.443 smtrat::sampling< S > Struct Template Reference . . . 1534
0.15.443.1 Member Function Documentation . . . 1534
0.15.444 smtrat::covering_ng::sampling< SamplingAlgorithm::LOWER_UPPER_BETWEEN_↩ SAMPLING > Struct Reference . . . 1534
0.15.444.1 Member Function Documentation . . . 1534
0.15.445 smtrat::sampling< SamplingAlgorithm::LOWER_UPPER_BETWEEN_SAMPLING > Struct Reference . . . 1534
0.15.445.1 Member Function Documentation . . . 1535
0.15.446 smtrat::SATModule< Settings > Class Template Reference . . . 1535
0.15.446.1 Detailed Description . . . 1541
0.15.446.2 Member Typedef Documentation . . . 1541
0.15.446.3 Member Enumeration Documentation . . . 1541
0.15.446.4 Constructor & Destructor Documentation . . . 1541
0.15.446.5 Member Function Documentation . . . 1542
0.15.446.6 Friends And Related Function Documentation . . . 1565
0.15.446.7 Field Documentation . . . 1565
0.15.447 smtrat::parser::ScriptParser< Callee > Struct Template Reference . . . 1567
0.15.447.1 Constructor & Destructor Documentation . . . 1568
0.15.447.2 Member Function Documentation . . . 1568
0.15.447.3 Field Documentation . . . 1568

0.15.448 smtrat::parser::ParserState::ScriptScope Struct Reference . . . 1569
0.15.448.1 Constructor & Destructor Documentation . . . 1569
0.15.448.2 Member Function Documentation . . . 1570
0.15.449 smtrat::mcsat::onecellcad::Section Struct Reference . . . 1570
0.15.449.1 Detailed Description . . . 1570
0.15.449.2 Field Documentation . . . 1570
0.15.450 smtrat::onecellcad::recursive::Section Struct Reference . . . 1570
0.15.450.1 Detailed Description . . . 1571
0.15.450.2 Field Documentation . . . 1571
0.15.451 smtrat::mcsat::onecellcad::levelwise::SectionHeuristic1 Struct Reference . . . 1571
0.15.451.1 Field Documentation . . . 1571
0.15.452 smtrat::mcsat::onecellcad::levelwise::SectionHeuristic2 Struct Reference . . . 1571
0.15.452.1 Field Documentation . . . 1571
0.15.453 smtrat::mcsat::onecellcad::levelwise::SectionHeuristic3 Struct Reference . . . 1571
0.15.453.1 Field Documentation . . . 1572
0.15.454 smtrat::mcsat::onecellcad::Sector Struct Reference . . . 1572
0.15.454.1 Detailed Description . . . 1572
0.15.454.2 Field Documentation . . . 1572
0.15.455 smtrat::onecellcad::recursive::Sector Struct Reference . . . 1572
0.15.455.1 Detailed Description . . . 1573
0.15.455.2 Member Function Documentation . . . 1573
0.15.455.3 Field Documentation . . . 1573
0.15.456 smtrat::mcsat::onecellcad::levelwise::SectorHeuristic1 Struct Reference . . . 1573
0.15.456.1 Field Documentation . . . 1573
0.15.457 smtrat::mcsat::onecellcad::levelwise::SectorHeuristic2 Struct Reference . . . 1573
0.15.457.1 Field Documentation . . . 1573
0.15.458 smtrat::mcsat::onecellcad::levelwise::SectorHeuristic3 Struct Reference . . . 1574
0.15.458.1 Field Documentation . . . 1574
0.15.459 smtrat::subtropical::Separator Struct Reference . . . 1574
0.15.459.1 Detailed Description . . . 1574
0.15.459.2 Constructor & Destructor Documentation . . . 1574
0.15.459.3 Field Documentation . . . 1574
0.15.460 smtrat::mcsat::SequentialAssignment< Backends > Struct Template Reference . . . 1575
0.15.461 smtrat::mcsat::SequentialExplanation< Backends > Struct Template Reference . . . 1575
0.15.462 benchmax::settings::Settings Struct Reference . . . 1575
0.15.462.1 Detailed Description . . . 1575
0.15.463 smtrat::settings::Settings Struct Reference . . . 1575
0.15.464 smtrat::SettingsComponents Class Reference . . . 1575
0.15.464.1 Member Function Documentation . . . 1575
0.15.465 benchmax::SettingsParser Class Reference . . . 1575
0.15.465.1 Detailed Description . . . 1575
0.15.466 smtrat::SettingsParser Class Reference . . . 1575

0.15.467 smtrat::CNFerModule::SettingsType Struct Reference . . . 1576
0.15.467.1 Field Documentation . . . 1576
0.15.468 smtrat::parser::SExpressionParser Struct Reference . . . 1576
0.15.468.1 Constructor & Destructor Documentation . . . 1576
0.15.468.2 Field Documentation . . . 1576
0.15.469 smtrat::parser::SExpressionSequence< T > Struct Template Reference . . . 1576
0.15.469.1 Constructor & Destructor Documentation . . . 1577
0.15.469.2 Field Documentation . . . 1577
0.15.470 smtrat::ShortFormulaEncoder Class Reference . . . 1577
0.15.470.1 Constructor & Destructor Documentation . . . 1577
0.15.470.2 Member Function Documentation . . . 1578
0.15.470.3 Field Documentation . . . 1578
0.15.471 benchmax::simple_parser Class Reference . . . 1578
0.15.471.1 Constructor & Destructor Documentation . . . 1579
0.15.471.2 Member Function Documentation . . . 1579
0.15.472 smtrat::parser::Theories::SimpleSortAdder Struct Reference . . . 1579
0.15.472.1 Detailed Description . . . 1579
0.15.472.2 Constructor & Destructor Documentation . . . 1580
0.15.472.3 Member Function Documentation . . . 1580
0.15.472.4 Field Documentation . . . 1580
0.15.473 smtrat::parser::SimpleSymbolParser Struct Reference . . . 1580
0.15.473.1 Detailed Description . . . 1580
0.15.473.2 Constructor & Destructor Documentation . . . 1580
0.15.473.3 Field Documentation . . . 1580
0.15.474 smtrat::expression::simplifier::Simplifier Class Reference . . . 1581
0.15.474.1 Member Function Documentation . . . 1581
0.15.475 smtrat::expression::simplifier::SimplifierChainCaller< chainID > Struct Template Reference . . . 1581
0.15.475.1 Member Function Documentation . . . 1581
0.15.475.2 Field Documentation . . . 1581
0.15.476 smtrat::expression::simplifier::SimplifierChainCaller< 0 > Struct Reference . . . 1581
0.15.476.1 Member Function Documentation . . . 1582
0.15.477 smtrat::expression::simplifier::SingletonSimplifier Struct Reference . . . 1582
0.15.477.1 Member Function Documentation . . . 1582
0.15.478 smtrat::cad::sample_compare::size Struct Reference . . . 1583
0.15.479 smtrat::parser::Skipper Struct Reference . . . 1583
0.15.479.1 Constructor & Destructor Documentation . . . 1584
0.15.479.2 Field Documentation . . . 1584
0.15.480 benchmax::SlurmBackend Class Reference . . . 1584
0.15.480.1 Detailed Description . . . 1584
0.15.480.2 Member Function Documentation . . . 1585
0.15.480.3 Field Documentation . . . 1585
0.15.481 benchmax::settings::SlurmBackendSettings Struct Reference . . . 1586

0.15.481.1 Detailed Description . . . 1586
0.15.481.2 Field Documentation . . . 1586
0.15.482 smtrat::parser::SMTLIBParser Class Reference . . . 1587
0.15.482.1 Constructor & Destructor Documentation . . . 1588
0.15.482.2 Member Function Documentation . . . 1588
0.15.482.3 Field Documentation . . . 1590
0.15.483 benchmax::SMTRAT Class Reference . . . 1590
0.15.483.1 Detailed Description . . . 1591
0.15.483.2 Constructor & Destructor Documentation . . . 1591
0.15.483.3 Member Function Documentation . . . 1591
0.15.483.4 Field Documentation . . . 1593
0.15.484 benchmax::SMTRAT_Analyzer Class Reference . . . 1593
0.15.484.1 Detailed Description . . . 1594
0.15.484.2 Constructor & Destructor Documentation . . . 1594
0.15.484.3 Member Function Documentation . . . 1594
0.15.484.4 Field Documentation . . . 1595
0.15.485 benchmax::SMTRAT_OPB Class Reference . . . 1595
0.15.485.1 Detailed Description . . . 1596
0.15.485.2 Constructor & Destructor Documentation . . . 1596
0.15.485.3 Member Function Documentation . . . 1596
0.15.485.4 Field Documentation . . . 1598
0.15.486 smtrat::execution::SoftAssertion Struct Reference . . . 1598
0.15.486.1 Field Documentation . . . 1598
0.15.487 smtrat::settings::SolverSettings Struct Reference . . . 1598
0.15.487.1 Field Documentation . . . 1599
0.15.488 smtrat::parser::SortedVariableParser Struct Reference . . . 1599
0.15.488.1 Constructor & Destructor Documentation . . . 1599
0.15.488.2 Field Documentation . . . 1599
0.15.489 smtrat::parser::SortParser Struct Reference . . . 1600
0.15.489.1 Constructor & Destructor Documentation . . . 1600
0.15.489.2 Member Function Documentation . . . 1600
0.15.490 smtrat::parser::SpecConstantParser Struct Reference . . . 1600
0.15.490.1 Constructor & Destructor Documentation . . . 1600
0.15.490.2 Field Documentation . . . 1601
0.15.491 smtrat::SplitSOSModule< Settings > Class Template Reference . . . 1601
0.15.491.1 Member Typedef Documentation . . . 1606
0.15.491.2 Member Enumeration Documentation . . . 1606
0.15.491.3 Constructor & Destructor Documentation . . . 1606
0.15.491.4 Member Function Documentation . . . 1607
0.15.491.5 Field Documentation . . . 1626
0.15.492 benchmax::SSHBackend Class Reference . . . 1628
0.15.492.1 Constructor & Destructor Documentation . . . 1629

0.15.492.2 Member Function Documentation . . . 1629
0.15.492.3 Field Documentation . . . 1630
0.15.493 benchmax::settings::SSHBackendSettings Struct Reference . . . 1630
0.15.493.1 Detailed Description . . . 1631
0.15.493.2 Field Documentation . . . 1631
0.15.494 benchmax::ssh::SSHConnection Class Reference . . . 1631
0.15.494.1 Detailed Description . . . 1632
0.15.494.2 Constructor & Destructor Documentation . . . 1632
0.15.494.3 Member Function Documentation . . . 1632
0.15.495 smtrat::vs::State Class Reference . . . 1633
0.15.495.1 Member Typedef Documentation . . . 1637
0.15.495.2 Member Enumeration Documentation . . . 1637
0.15.495.3 Constructor & Destructor Documentation . . . 1637
0.15.495.4 Member Function Documentation . . . 1638
0.15.496 smtrat::StaticUnionFind Struct Reference . . . 1657
0.15.496.1 Member Typedef Documentation . . . 1657
0.15.496.2 Member Function Documentation . . . 1658
0.15.497 smtrat::statistics::StatisticsSettings Struct Reference . . . 1658
0.15.497.1 Field Documentation . . . 1658
0.15.498 smtrat::StrategyGraph Class Reference . . . 1658
0.15.498.1 Constructor & Destructor Documentation . . . 1659
0.15.498.2 Member Function Documentation . . . 1659
0.15.498.3 Friends And Related Function Documentation . . . 1659
0.15.499 Minisat::StringOption Class Reference . . . 1660
0.15.499.1 Constructor & Destructor Documentation . . . 1660
0.15.499.2 Member Function Documentation . . . 1660
0.15.499.3 Field Documentation . . . 1661
0.15.500 smtrat::parser::StringParser Struct Reference . . . 1661
0.15.500.1 Detailed Description . . . 1661
0.15.500.2 Constructor & Destructor Documentation . . . 1661
0.15.500.3 Field Documentation . . . 1661
0.15.501 smtrat::STropModule< Settings > Class Template Reference . . . 1661
0.15.501.1 Member Typedef Documentation . . . 1667
0.15.501.2 Member Enumeration Documentation . . . 1667
0.15.501.3 Constructor & Destructor Documentation . . . 1667
0.15.501.4 Member Function Documentation . . . 1667
0.15.501.5 Field Documentation . . . 1685
0.15.502 benchmax::slurm::SubmitfileProperties Struct Reference . . . 1688
0.15.502.1 Detailed Description . . . 1688
0.15.502.2 Field Documentation . . . 1688
0.15.503 smtrat::vs::Substitution Class Reference . . . 1689
0.15.503.1 Member Enumeration Documentation . . . 1690

0.15.503.2 Constructor & Destructor Documentation . . . 1690
0.15.503.3 Member Function Documentation . . . 1691
0.15.504 smtrat::vs::substitutionPointerEqual Struct Reference . . . 1693
0.15.504.1 Member Function Documentation . . . 1693
0.15.505 smtrat::vs::substitutionPointerHash Struct Reference . . . 1693
0.15.505.1 Member Function Documentation . . . 1693
0.15.506 smtrat::cadcells::datastructures::SymbolicInterval Class Reference . . . 1693
0.15.506.1 Detailed Description . . . 1694
0.15.506.2 Constructor & Destructor Documentation . . . 1694
0.15.506.3 Member Function Documentation . . . 1694
0.15.507 smtrat::parser::SymbolParser Struct Reference . . . 1695
0.15.507.1 Constructor & Destructor Documentation . . . 1695
0.15.507.2 Field Documentation . . . 1695
0.15.508 smtrat::SymmetryModule< Settings > Class Template Reference . . . 1696
0.15.508.1 Member Typedef Documentation . . . 1701
0.15.508.2 Member Enumeration Documentation . . . 1701
0.15.508.3 Constructor & Destructor Documentation . . . 1701
0.15.508.4 Member Function Documentation . . . 1701
0.15.508.5 Field Documentation . . . 1719
0.15.509 smtrat::lra::Tableau< Settings, T1, T2 > Class Template Reference . . . 1722
0.15.509.1 Constructor & Destructor Documentation . . . 1724
0.15.509.2 Member Function Documentation . . . 1724
0.15.510 smtrat::lra::TableauEntry< T1, T2 > Class Template Reference . . . 1740
0.15.510.1 Constructor & Destructor Documentation . . . 1741
0.15.510.2 Member Function Documentation . . . 1741
0.15.511 smtrat::cadcells::datastructures::TaggedIndexedRoot Struct Reference . . . 1743
0.15.511.1 Field Documentation . . . 1744
0.15.512 smtrat::mcsat::onecellcad::TagPoly Struct Reference . . . 1744
0.15.512.1 Detailed Description . . . 1744
0.15.512.2 Field Documentation . . . 1744
0.15.513 smtrat::Task Class Reference . . . 1744
0.15.513.1 Constructor & Destructor Documentation . . . 1745
0.15.513.2 Member Function Documentation . . . 1745
0.15.514 smtrat::parser::TermParser Struct Reference . . . 1745
0.15.514.1 Member Typedef Documentation . . . 1746
0.15.514.2 Constructor & Destructor Documentation . . . 1746
0.15.514.3 Field Documentation . . . 1746
0.15.515 smtrat::mcsat::vs::helper::TestCandidate Struct Reference . . . 1747
0.15.515.1 Member Function Documentation . . . 1747
0.15.515.2 Field Documentation . . . 1747
0.15.516 smtrat::parser::Theories Struct Reference . . . 1747
0.15.516.1 Detailed Description . . . 1748

0.15.516.2 Member Typedef Documentation . . . 1748
0.15.516.3 Constructor & Destructor Documentation . . . 1748
0.15.516.4 Member Function Documentation . . . 1749
0.15.517 smtrat::parser::TheoryError Struct Reference . . . 1750
0.15.517.1 Member Function Documentation . . . 1751
0.15.517.2 Friends And Related Function Documentation . . . 1751
0.15.518 smtrat::mcsat::TheoryLevel Struct Reference . . . 1751
0.15.518.1 Field Documentation . . . 1751
0.15.519 smtrat::Module::TheoryPropagation Struct Reference . . . 1752
0.15.519.1 Constructor & Destructor Documentation . . . 1752
0.15.519.2 Member Function Documentation . . . 1752
0.15.519.3 Field Documentation . . . 1753
0.15.520 smtrat::TheoryVarSchedulerStatic< vot > Class Template Reference . . . 1753
0.15.520.1 Detailed Description . . . 1754
0.15.520.2 Constructor & Destructor Documentation . . . 1754
0.15.520.3 Member Function Documentation . . . 1754
0.15.520.4 Field Documentation . . . 1755
0.15.521 smtrat::ThreadPool Class Reference . . . 1757
0.15.521.1 Constructor & Destructor Documentation . . . 1757
0.15.521.2 Member Function Documentation . . . 1757
0.15.522 smtrat::cad::debug::TikzBasePrinter Class Reference . . . 1757
0.15.522.1 Member Function Documentation . . . 1758
0.15.522.2 Field Documentation . . . 1758
0.15.523 smtrat::cad::debug::TikzDAGPrinter Class Reference . . . 1758
0.15.523.1 Member Function Documentation . . . 1759
0.15.523.2 Field Documentation . . . 1760
0.15.524 smtrat::cad::debug::TikzHistoryPrinter Class Reference . . . 1760
0.15.524.1 Member Function Documentation . . . 1760
0.15.525 smtrat::cad::debug::TikzTreePrinter Class Reference . . . 1761
0.15.525.1 Member Function Documentation . . . 1761
0.15.525.2 Field Documentation . . . 1762
0.15.526 benchmax::Tool Class Reference . . . 1762
0.15.526.1 Detailed Description . . . 1763
0.15.526.2 Constructor & Destructor Documentation . . . 1764
0.15.526.3 Member Function Documentation . . . 1764
0.15.526.4 Friends And Related Function Documentation . . . 1765
0.15.526.5 Field Documentation . . . 1765
0.15.527 benchmax::settings::ToolSettings Struct Reference . . . 1766
0.15.527.1 Detailed Description . . . 1766
0.15.527.2 Field Documentation . . . 1766
0.15.528 smtrat::TotalizerEncoder Class Reference . . . 1767
0.15.528.1 Detailed Description . . . 1768

0.15.528.2 Constructor & Destructor Documentation . . . 1768
0.15.528.3 Member Function Documentation . . . 1768
0.15.528.4 Field Documentation . . . 1769
0.15.529 smtrat::TotalizerTree Class Reference . . . 1769
0.15.529.1 Constructor & Destructor Documentation . . . 1769
0.15.529.2 Member Function Documentation . . . 1769
0.15.530 smtrat::covering_ng::formula::node_ds::TRUE Struct Reference . . . 1769
0.15.531 smtrat::cad::projection_compare::type Struct Reference . . . 1769
0.15.532 smtrat::cad::sample_compare::type Struct Reference . . . 1769
0.15.533 smtrat::UFCegarModule< Settings > Class Template Reference . . . 1770
0.15.533.1 Member Typedef Documentation . . . 1775
0.15.533.2 Member Enumeration Documentation . . . 1775
0.15.533.3 Constructor & Destructor Documentation . . . 1775
0.15.533.4 Member Function Documentation . . . 1775
0.15.533.5 Field Documentation . . . 1795
0.15.534 smtrat::expression::UnaryExpression Struct Reference . . . 1797
0.15.534.1 Constructor & Destructor Documentation . . . 1797
0.15.534.2 Field Documentation . . . 1797
0.15.535 smtrat::cad::debug::UnifiedData Struct Reference . . . 1797
0.15.535.1 Member Function Documentation . . . 1798
0.15.535.2 Field Documentation . . . 1798
0.15.536 smtrat::parser::types::UninterpretedTheory Struct Reference . . . 1798
0.15.536.1 Detailed Description . . . 1798
0.15.536.2 Member Typedef Documentation . . . 1798
0.15.537 smtrat::parser::UninterpretedTheory Struct Reference . . . 1799
0.15.537.1 Detailed Description . . . 1800
0.15.537.2 Constructor & Destructor Documentation . . . 1800
0.15.537.3 Member Function Documentation . . . 1800
0.15.537.4 Field Documentation . . . 1801
0.15.538 smtrat::UnionFindInterface< T, Implementation > Struct Template Reference . . . 1802
0.15.538.1 Member Typedef Documentation . . . 1802
0.15.538.2 Constructor & Destructor Documentation . . . 1802
0.15.538.3 Member Function Documentation . . . 1803
0.15.539 smtrat::UnionFindModule< Settings > Class Template Reference . . . 1803
0.15.539.1 Member Typedef Documentation . . . 1808
0.15.539.2 Member Enumeration Documentation . . . 1809
0.15.539.3 Constructor & Destructor Documentation . . . 1809
0.15.539.4 Member Function Documentation . . . 1809
0.15.539.5 Field Documentation . . . 1828
0.15.540 smtrat::SATModule< Settings >::UnorderedClauseLookup::UnorderedClauseHasher Struct Reference . . . 1830
0.15.540.1 Member Function Documentation . . . 1830

0.15.541 smtrat::UnsatCore< Solver, Strategy > Class Template Reference 1830
0.15.541.1 Constructor & Destructor Documentation 1831
0.15.541.2 Member Function Documentation 1831
0.15.542 smtrat::unsatcore::UnsatCoreBackend< Solver, Strategy > Class Template Reference . . 1831
0.15.543 smtrat::unsatcore::UnsatCoreBackend< Solver, UnsatCoreStrategy::ModelExclusion > Class Template Reference 1831
0.15.543.1 Detailed Description 1832
0.15.543.2 Constructor & Destructor Documentation 1832
0.15.543.3 Member Function Documentation 1832
0.15.544 smtrat::vs::unsignedTripleCmp Struct Reference 1832
0.15.544.1 Member Function Documentation 1832
0.15.545 smtrat::parser::UserFunctionInstantiator Struct Reference 1833
0.15.545.1 Constructor & Destructor Documentation 1833
0.15.545.2 Member Function Documentation 1833
0.15.545.3 Field Documentation 1834
0.15.546 smtrat::validation::ValidationCollector Class Reference 1834
0.15.546.1 Member Function Documentation 1834
0.15.547 smtrat::validation::ValidationPoint Class Reference 1834
0.15.547.1 Member Function Documentation 1835
0.15.548 smtrat::validation::ValidationPrinter< SOF > Struct Template Reference 1835
0.15.549 smtrat::validation::ValidationSettings Struct Reference 1835
0.15.549.1 Member Function Documentation 1836
0.15.549.2 Field Documentation 1836
0.15.550 smtrat::lra::Value< T > Class Template Reference 1836
0.15.550.1 Constructor & Destructor Documentation 1837
0.15.550.2 Member Function Documentation 1838
0.15.551 smtrat::lra::Variable< T1, T2 > Class Template Reference 1843
0.15.551.1 Constructor & Destructor Documentation 1845
0.15.551.2 Member Function Documentation 1846
0.15.552 smtrat::vb::Variable< T > Class Template Reference 1854
0.15.552.1 Detailed Description 1855
0.15.552.2 Constructor & Destructor Documentation 1855
0.15.552.3 Member Function Documentation 1855
0.15.553 smtrat::vb::VariableBounds< T > Class Template Reference 1857
0.15.553.1 Detailed Description 1858
0.15.553.2 Member Typedef Documentation 1858
0.15.553.3 Constructor & Destructor Documentation 1858
0.15.553.4 Member Function Documentation 1859
0.15.554 smtrat::VariableCapsule Class Reference 1864
0.15.554.1 Constructor & Destructor Documentation 1864
0.15.554.2 Member Function Documentation 1864
0.15.554.3 Field Documentation 1865

0.15.555 smtrat::mcsat::variableordering::VariableIDs Struct Reference 1865
0.15.555.1 Member Function Documentation 1865
0.15.555.2 Field Documentation 1865
0.15.556 smtrat::VariableRewriteRule Class Reference 1865
0.15.556.1 Constructor & Destructor Documentation 1866
0.15.556.2 Field Documentation 1866
0.15.557 smtrat::parser::conversion::VariantConverter< Res > Struct Template Reference 1866
0.15.557.1 Detailed Description 1867
0.15.557.2 Member Typedef Documentation 1867
0.15.557.3 Member Function Documentation 1867
0.15.557.4 Field Documentation 1867
0.15.558 smtrat::VariantMap< Key, Value > Class Template Reference 1868
0.15.558.1 Detailed Description 1868
0.15.558.2 Member Function Documentation 1868
0.15.558.3 Field Documentation 1870
0.15.559 smtrat::parser::conversion::VariantVariantConverter< Res > Struct Template Reference . 1870
0.15.559.1 Detailed Description 1870
0.15.559.2 Member Typedef Documentation 1870
0.15.559.3 Member Function Documentation 1870
0.15.560 smtrat::VarSchedulerBase Class Reference 1871
0.15.560.1 Detailed Description 1871
0.15.560.2 Constructor & Destructor Documentation 1872
0.15.560.3 Member Function Documentation 1872
0.15.560.4 Field Documentation 1873
0.15.561 smtrat::VarSchedulerFixedRandom Class Reference 1874
0.15.561.1 Constructor & Destructor Documentation 1875
0.15.561.2 Member Function Documentation 1875
0.15.561.3 Field Documentation 1876
0.15.562 smtrat::VarSchedulerMcsatActivityPreferTheory< vot > Class Template Reference 1877
0.15.562.1 Detailed Description 1878
0.15.562.2 Constructor & Destructor Documentation 1878
0.15.562.3 Member Function Documentation 1878
0.15.562.4 Field Documentation 1879
0.15.563 smtrat::VarSchedulerMcsatBase Class Reference 1880
0.15.563.1 Detailed Description 1881
0.15.563.2 Constructor & Destructor Documentation 1881
0.15.563.3 Member Function Documentation 1881
0.15.563.4 Field Documentation 1882
0.15.564 smtrat::VarSchedulerMcsatBooleanFirst< vot > Class Template Reference 1884
0.15.564.1 Detailed Description 1885
0.15.564.2 Constructor & Destructor Documentation 1885
0.15.564.3 Member Function Documentation 1885

0.15.564.4 Field Documentation . . . 1886
0.15.565 smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler > Class Template Reference . 1887
0.15.565.1 Detailed Description . . . 1888
0.15.565.2 Constructor & Destructor Documentation . . . 1888
0.15.565.3 Member Function Documentation . . . 1888
0.15.565.4 Field Documentation . . . 1889
0.15.566 smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities > Class Template Reference . . . 1891
0.15.566.1 Detailed Description . . . 1892
0.15.566.2 Constructor & Destructor Documentation . . . 1892
0.15.566.3 Member Function Documentation . . . 1892
0.15.566.4 Field Documentation . . . 1893
0.15.567 smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot > Class Template Reference . . . 1895
0.15.567.1 Detailed Description . . . 1896
0.15.567.2 Constructor & Destructor Documentation . . . 1896
0.15.567.3 Member Function Documentation . . . 1896
0.15.567.4 Field Documentation . . . 1897
0.15.568 smtrat::VarSchedulerMinisat Class Reference . . . 1898
0.15.568.1 Detailed Description . . . 1899
0.15.568.2 Constructor & Destructor Documentation . . . 1899
0.15.568.3 Member Function Documentation . . . 1899
0.15.568.4 Field Documentation . . . 1900
0.15.569 smtrat::VarSchedulerRandom Class Reference . . . 1902
0.15.569.1 Detailed Description . . . 1902
0.15.569.2 Constructor & Destructor Documentation . . . 1902
0.15.569.3 Member Function Documentation . . . 1903
0.15.569.4 Field Documentation . . . 1904
0.15.570 smtrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic > Class Template Reference . . . 1905
0.15.570.1 Detailed Description . . . 1905
0.15.570.2 Constructor & Destructor Documentation . . . 1906
0.15.570.3 Member Function Documentation . . . 1906
0.15.570.4 Field Documentation . . . 1907
0.15.571 Minisat::vec< T > Class Template Reference . . . 1908
0.15.571.1 Constructor & Destructor Documentation . . . 1909
0.15.571.2 Member Function Documentation . . . 1909
0.15.572 smtrat::parser::conversion::VectorVariantConverter< Res > Struct Template Reference . 1911
0.15.572.1 Detailed Description . . . 1911
0.15.572.2 Member Typedef Documentation . . . 1911
0.15.572.3 Member Function Documentation . . . 1912
0.15.573 smtrat::subtropical::Vertex Struct Reference . . . 1912
0.15.573.1 Detailed Description . . . 1912

0.15.573.2 Constructor & Destructor Documentation . . . 1912
0.15.573.3 Member Function Documentation . . . 1912
0.15.573.4 Field Documentation . . . 1913
0.15.574 smtrat::VSModule< Settings > Class Template Reference . . . 1913
0.15.574.1 Member Typedef Documentation . . . 1918
0.15.574.2 Member Enumeration Documentation . . . 1918
0.15.574.3 Constructor & Destructor Documentation . . . 1918
0.15.574.4 Member Function Documentation . . . 1919
0.15.574.5 Field Documentation . . . 1940
0.15.575 Minisat::Watcher Struct Reference . . . 1942
0.15.575.1 Detailed Description . . . 1942
0.15.575.2 Constructor & Destructor Documentation . . . 1943
0.15.575.3 Member Function Documentation . . . 1943
0.15.575.4 Friends And Related Function Documentation . . . 1943
0.15.575.5 Field Documentation . . . 1943
0.15.576 smtrat::ICPModule< Settings >::weights Struct Reference . . . 1943
0.15.576.1 Field Documentation . . . 1943
0.15.577 benchmax::XMLWriter Class Reference . . . 1944
0.15.577.1 Detailed Description . . . 1944
0.15.577.2 Constructor & Destructor Documentation . . . 1944
0.15.577.3 Member Function Documentation . . . 1944
0.15.578 smtrat::covering_ng::formula::node_ds::XOR Struct Reference . . . 1944
0.15.578.1 Field Documentation . . . 1944
0.15.579 benchmax::Z3 Class Reference . . . 1944
0.15.579.1 Detailed Description . . . 1945
0.15.579.2 Constructor & Destructor Documentation . . . 1945
0.15.579.3 Member Function Documentation . . . 1945
0.15.579.4 Field Documentation . . . 1946

0.1 SMT-RAT

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

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

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

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

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

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

0.2 Installation

0.2.1 Requirements

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

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

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

Optional dependencies

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

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

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

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

- `g++` >= 11
- `clang` >= 13

0.2.2 Download

Here are archived versions of SMT-RAT for download:

- latest

0.2.3 Configuration

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

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

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

0.2.4 Build

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

```
$ make smtrat-shared
```

Relevant targets you may want to build individually include:

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

0.2.5 Check build

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

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

0.3 System architecture

0.3.1 Different libraries

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

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

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

0.3.2 Software design

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

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

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

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

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

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

0.3.3 Modules

A module m holds a set of formulas, called its set of received formulas and denoted by $C_{rcv}(m)$. The main function of a module is `check(bool full)`, which either decides whether $C_{rcv}(m)$ is satisfiable or not, returning `SAT` or `UNSAT`, respectively, or returns `UNKNOWN`. A set of formulas is semantically defined by their conjunction. If the function's argument `full` is set to `false`, the underlying procedure of m is allowed to omit hard obstacles during solving at the cost of returning `UNKNOWN` in more cases. We can manipulate $C_{rcv}(m)$ by adding (removing) formulas φ to (from) it with `add(\f$\varphi\f$)` (`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.

0.3.4 Strategy

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

0.3.5 Manager

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

0.3.6 Procedures implemented as modules

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

0.3.7 Infeasible subsets and lemmas

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

0.4 Modules

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

0.4.1 Main members of a module

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

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

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

0.4.2 Interfaces to implement

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

0.4.2.1 Informing about a constraint

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

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

0.4.2.2 Adding a received formula

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

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

0.4.2.3 Removing a received formula

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

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

0.4.2.4 Checking for satisfiability

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

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

0.4.2.5 Updating the model/satisfying assignment

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

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

0.4.3 Running backend modules

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

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

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

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

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

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

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

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

Adds the formula at the given posititon 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 `add↩SubformulaToPassedFormula`), the next call of `removeSubformula` of this module removes this formula from the passed formula. Specifying received formulas being the origins of a passed formula highly improves the incremental solving performance, so we recommend to do so.

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

0.4.4 Auxiliary functions

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

0.4.5 Existing module implementation

Available modules

0.4.6 Available modules

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

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

0.4.6.1 BEModule

0.4.6.2 BVModule

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

0.4.6.3 CNFerModule

Transforms its received formula into conjunctive normal form CNF.

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

0.4.6.4 CSplitModule

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

0.4.6.5 CoCoAGBModule

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

0.4.6.6 CubeLIAModule

Implements cube-based tests for linear integer arithmetic based on **?**.

0.4.6.7 CurryModule

Implements curryfiyng preprcessing as described in **?**.

0.4.6.8 EMModule

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

0.4.6.9 ESModule

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

0.4.6.10 FPPModule

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

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

0.4.6.11 FouMoModule

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

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

0.4.6.12 GBModule

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

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

0.4.6.13 GBPPModule

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

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

0.4.6.14 ICEModule

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

0.4.6.15 ICPModule

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

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

0.4.6.16 IncWidthModule

This module is meant to be used for solving nonlinear integer arithmetic problems by encoding them into bitvector arithmetic formulas (as done in 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.

0.4.6.17 IntBlastModule

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

0.4.6.18 IntEqModule

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

0.4.6.19 LRAModule

Implements the SMT compliant simplex method presented in **?**. Hence, this module can decide the consistency of any conjunction consisting only of linear real arithmetic constraints. Furthermore, it might also find the consistency of a conjunction of constraints even if they are not all linear and calls a backend after removing some redundant linear constraints, if the linear constraints are satisfiable and the found solution does not satisfy the non-linear constraints. Note that the 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.

0.4.6.19.0.1 Integer arithmetic In order to find integer solutions, this module applies, depending on which settings are used, branch-and-bound, the construction of Gomory cuts and the generation of cuts from proofs **?**. It is also supported to combine these approaches. Note that for all of them the 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.

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

0.4.6.20 LVEModule

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

0.4.6.21 MCBModule

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

0.4.6.22 NRAILModule

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

0.4.6.23 NewCADModule

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

0.4.6.24 PBGaussModule

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

0.4.6.25 PBPPModule

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

0.4.6.26 PFEModule

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

Given a constraint $p \sim 0$ with $\sim \in \{=, \neq, \geq, >, \leq, <\}$ and bounds b on the variables in p, the module does the following: It computes a factor q such that $p = q \cdot r$ and evaluates q on the intervals represented by the bounds. If the resulting interval $q(b)$ 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.

0.4.6.26.0.1 Simplifications

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

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

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

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

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

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

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

0.4.6.27 SATModule

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

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

0.4.6.28 STropModule

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

0.4.6.29 SplitSOSModule

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

0.4.6.30 SymmetryModule

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

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

0.4.6.31 UFCegarModule

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

0.4.6.32 UnionFindModule

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

0.4.6.33 VSModule

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

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

0.5 Strategies

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

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

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

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

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

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

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

0.6 Using SMT-RAT

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

0.6.1 Standalone solver

Before actually compiling SMT-RAT into a binary, an appropriate strategy should be selected. While a number of strategies are available, it sometimes makes sense to craft a strategy for the specific problem at hand. Please refer to Strategies on how to create strategies. To select a strategy use `ccmake` to set the `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
```

0.6.1.1 Formula analysis

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

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

0.6.1.2 Preprocessing

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

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

0.6.1.3 Quantifier elimination

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

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

0.6.1.4 DIMACS solving

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

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

0.6.1.5 Pseudo-Boolean solving

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

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

0.6.1.6 Optimization

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

0.6.2 Embedding in other software

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

0.6.2.1 Interface

The easiest way is to embed a certain strategy.

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

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

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

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

- `std::vector<FormulaT> lemmas() const` Returns valid formulas, which we call lemmas. For instance the `ICPModule` might return lemmas being splitting decisions, which need to be processed in, e.g., a SAT solver. A *splitting decision* has in general the form `(c_1 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.

0.6.2.2 Syntax of formulas

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

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

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.

0.6.2.3 Boolean combinations of constraints and Boolean variables

For more information, check out the docs of `CArL`.

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

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

We give a small example constructing the formula

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

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

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

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

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

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

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

```
carl::Sort sortS = smtrat::newSort( "S" );
carl::Sort sortT = smtrat::newSort( "T" );
carl::UVariable uu( u, sortS );
carl::UVariable uv( v, sortT );
carl::UVariable uw( w, sortS );
carl::UninterpretedFunction f = smtrat::newUF( "f", sortS, sortS, sortT );
carl::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.

0.6.2.4 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 $a_1, \ldots, a_n$,

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

and

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

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

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

0.6.2.5 Linking

[Todo: example how linking works]

0.7 Other tools

0.7.1 Benchmarking

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

0.7.2 Delta debugging

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

Note: Our legacy delta tool has been moved to `its own repository`.

0.7.2.1 Using ddSMT

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

For a complete guide, we refer to the `documentation of ddSMT`.

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

First, install `ddSMT` and `z3`:
```
pip3 install ddsmt
sudo apt install z3
```

In case of a *failing assertion* on `in_file.smt2`, run
```
~/src/smtrat/utilities/ddsmt/ddsmt-assertion in_file.smt2 out_file.smt2 path_to_solver
```

In case SMT-RAT returns a *wrong result* on `in_file.smt2`, run
```
~/src/smtrat/utilities/ddsmt/ddsmt-wrong in_file.smt2 out_file.smt2 path_to_solver
```

0.7.3 Analyzer

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

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

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

0.7.4 Preprocessing

WIP

0.7.5 Benchmax

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

0.7.5.1 General usage

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

This will run benchmax with the SMT-RAT tool at `/path/to/smtrat-static` on the `/path/to/benchmark/set` with a time limit of 1 minute and a memory limit of 4 gigabyte. The 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`.

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

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

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

0.7.5.1.3 Using a solver with options To pass options to the tested tool, use a string consisting of the path to the tool and the required options instead of only the path. In the example above, if we wanted to use the SMT-RAT tool at `/path/to/smtrat-static` with some option `--op` and argument `arg`, we would call

```
./benchmax -T 1m -M 4Gi -S "/path/to/smtrat-static -op arg" -X ...
```

(`...` being the same as before).

0.7.5.2 Tools

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

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

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

- benchmax::MathSAT
- benchmax::Minisatp
- benchmax::SMTRAT
- benchmax::SMTRAT_OPB
- benchmax::Z3

0.7.5.3 Backends

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

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

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

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

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

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

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

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

0.7.5.4 Troubleshooting

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

0.7.5.5 Benchmax python utility

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

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

Dependencies:

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

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

0.7.5.5.1 Loading XMLs into a pandas dataframe First, install this directory as python library; e.g. on Ubuntu

```
cd ~/.local/lib/python3.8/site-packages/ # path to your python site-packages directory
ln -s ~/src/smtrat/utilities/benchmax/ # path to the benchmax utility
```

In your Jupyter notebook:

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

or, to load 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).

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

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

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

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

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

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

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

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

0.7.5.5.5 Exporting data to Latex

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

0.7.5.5.5.2 matplotlib to tikz plot see `tikzplotlib`

0.8 Developers information

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

0.8.1 Code style

Please follow the code style rules from CArL documentation.

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

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

0.8.2 Documentation

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

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

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

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

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

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

0.8.2.1 Code comments

0.8.2.1.1 File headers

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

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

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

0.8.2.1.3 Class headers

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

0.8.2.1.4 Method headers

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

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

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

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

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

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

0.8.2.2 Writing out-of-source documentation

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

0.8.3 Settings

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

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

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

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

0.8.4 Logging

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

It provides the following macros for logging:

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

Where the arguments mean the following:

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

Typically, logging looks like this:

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

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

0.8.5 Statistics and timing

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

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

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

```
#pragma once
#include <smtrat-common/statistics/Statistics.h>
#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
```

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

This is then instantiated by calling

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

or, as shortcut

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

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

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

0.8.5.0.2 Timing The statistics framework has the ability to easily collect timings.

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

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

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

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

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

The measured timings will then appear alongside the statistics.

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

0.8.6 Testing

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

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

0.8.7 Validation

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

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

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

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

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

Hint: to put it in a static variable, use

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

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

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

To combine the two steps above, use:

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

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

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

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

0.8.8 Checkpoints

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

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

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

0.8.8.0.2 Specyfing checkpoints Insert

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

into the code.

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

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

or

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

to assert that a checkpoint is reached.

Afterwards, run the respective SMT-RAT code.

You can clear all checkpoints of a channel using

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

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

0.8.9 Finding and Reporting Bugs

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

- SMT-RAT crashes.
- A library used through SMT-RAT crashes.
- SMT-RAT gives incorrect results.
- SMT-RAT does not terminate (for reasonably sized inputs).
- SMT-RAT does not provide a method or functionality that should be available according to this documentation.
- SMT-RAT does not provide a method or functionality that you consider crucial or trivial for some of the 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/ths-rwth/smtrat/issues.

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

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

0.9 README

This is dead code.

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

0.10 NewCoveringModule #and

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

0.10.1 Introduction

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

0.10.2 Usage

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

0.10.3 Efficiency

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

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

0.11 Todo List

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

Check what can be checked here.

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

check that definition matches the sort

0.12 Hierarchical Index

0.12.1 Class Hierarchy

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

smtrat::AbstractModuleFactory . . . 232
- smtrat::ModuleFactory< Module > . . . 1193

smtrat::parser::AbstractTheory . . . 232
- smtrat::parser::ArithmeticTheory . . . 241
- smtrat::parser::BitvectorTheory . . . 310
- smtrat::parser::BooleanEncodingTheory . . . 317
- smtrat::parser::CoreTheory . . . 478
- smtrat::parser::UninterpretedTheory . . . 1799

smtrat::cad::sample_compare::absvalue . . . 235
smtrat::analyzer::AnalysisSettings . . . 235
smtrat::covering_ng::formula::node_ds::AND . . . 236
smtrat::AnnotatedBVTerm . . . 236
smtrat::cadcells::representation::approximation::ApxSettings . . . 238
benchmax::slurm::ArchiveProperties . . . 240
smtrat::parser::types::ArithmeticTheory . . . 244
smtrat::execution::Assertion . . . 244
smtrat::cad::preprocessor::AssignmentCollector . . . 244
smtrat::mcsat::arithmetic::AssignmentFinder . . . 246
smtrat::mcsat::smtaf::AssignmentFinder< Settings > . . . 246
smtrat::mcsat::arithmetic::AssignmentFinder_ctx . . . 246
smtrat::mcsat::arithmetic::AssignmentFinder_detail . . . 247
smtrat::mcsat::smtaf::AssignmentFinder_SMT . . . 248
smtrat::cadcells::datastructures::detail::AssignmentProperties . . . 249
smtrat::parser::Attribute . . . 249
smtrat::AxiomFactory . . . 252
benchmax::Backend . . . 253
- benchmax::CondorBackend . . . 459
- benchmax::LocalBackend . . . 1033
- benchmax::SSHBackend . . . 1628
- benchmax::SlurmBackend . . . 1584

smtrat::Backend< Settings > . . . 255
smtrat::BackendLink . . . 257
smtrat::BackendSynchronisation . . . 258
smtrat::Backtrackable< UnionFind > . . . 259
smtrat::mcsat::onecell::Base . . . 260
- smtrat::mcsat::onecell::BCFilteredSettings . . . 275
  - smtrat::mcsat::onecell::BCFilteredAllSelectiveSettings . . . 273
  - smtrat::mcsat::onecell::BCFilteredAllSettings . . . 274
  - smtrat::mcsat::onecell::BCFilteredBoundsSettings . . . 274
  - smtrat::mcsat::onecell::BCFilteredSamplesSettings . . . 275
- smtrat::mcsat::onecell::BCSettings . . . 276
  - smtrat::mcsat::onecell::BCApproximationSettings . . . 273
- smtrat::mcsat::onecell::LDBFilteredAllSelectiveSettings . . . 1017
- smtrat::mcsat::onecell::LDBSettings . . . 1017

smtrat::cadcells::datastructures::BaseDerivation< Properties > . . . 261
smtrat::cad::BaseProjection< Settings > . . . 263
- smtrat::cad::ModelBasedProjection< ProjectionSettings > . . . 1156
- smtrat::cad::Projection< Settings > . . . 1446
- smtrat::cad::ModelBasedProjection< incrementality, backtracking, Settings > . . . 1156
- smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings > . . . 1161
- smtrat::cad::Projection< incrementality, backtracking, Settings > . . . 1446
- smtrat::cad::Projection< Incrementality::FULL, BT, Settings > . . . 1463

smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings > . . . 1457
smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings > . . . 1468
smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings > . . . 1474
smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings > . . . 1480
smtrat::qe::cad::Projection< Settings > . . . 1451
smtrat::cad::BaseSettings . . . 268
smtrat::qe::cad::CADSettings . . . 382
smtrat::cad::Origin::BaseType . . . 271
benchmax::BenchmarkResult . . . 302
benchmax::BenchmarkSet . . . 304
benchmax::settings::BenchmarkSettings . . . 305
smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName > . . . 306
smtrat::Bimap< Expansion, const carl::Variable, &Expansion::mRationalization, const carl::Variable, &Expansion::mDiscretization > . . . 306
smtrat::Bimap< Linearization, const Poly, &Linearization::mNormalization, const Poly, &Linearization::m↩ Linearization > . . . 306
smtrat::expression::BinaryExpression . . . 308
smtrat::parser::types::BitvectorTheory . . . 312
smtrat::BlastedConstr . . . 313
smtrat::BlastedPoly . . . 314
smtrat::mcsat::Bookkeeping . . . 315
smtrat::covering_ng::formula::node_ds::BOOL . . . 317
smtrat::cadcells::datastructures::Bound . . . 321
smtrat::lra::Bound< T1, T2 > . . . 322
smtrat::mcsat::fm::Bound . . . 331
smtrat::vb::Bound< T > . . . 332
smtrat::Branching . . . 337
smtrat::BVAnnotation . . . 338
smtrat::BVDirectEncoder . . . 340
smtrat::cad::CAD< Settings > . . . 368
smtrat::qe::cad::CAD< Settings > . . . 371
smtrat::qe::cad::CAD< smtrat::qe::cad::CADSettings > . . . 371
smtrat::cad::CADConstraints< BT > . . . 373
smtrat::cad::CADConstraints< ProjectionSettings::backtracking > . . . 373
smtrat::cad::CADConstraints< Settings::backtracking > . . . 373
smtrat::cad::CADCore< CH > . . . 377
smtrat::cad::CADCore< CoreHeuristic::BySample > . . . 377
smtrat::cad::CADCore< CoreHeuristic::EnumerateAll > . . . 378
smtrat::cad::CADCore< CoreHeuristic::Interleave > . . . 378
smtrat::cad::CADCore< CoreHeuristic::PreferProjection > . . . 379
smtrat::cad::CADCore< CoreHeuristic::PreferSampling > . . . 379
smtrat::qe::cad::CADElimination . . . 379
smtrat::cad::CADPreprocessor . . . 380
smtrat::cad::CADPreprocessorSettings . . . 381
smtrat::cadcells::representation::cell< H > . . . 384
smtrat::cadcells::representation::cell< CellHeuristic::BIGGEST_CELL > . . . 385
smtrat::cadcells::representation::cell< CellHeuristic::BIGGEST_CELL_APPROXIMATION > . . . 385
smtrat::cadcells::representation::cell< CellHeuristic::BIGGEST_CELL_EW > . . . 385
smtrat::cadcells::representation::cell< CellHeuristic::CHAIN_EQ > . . . 386
smtrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS > . . . 386
smtrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS_EQ > . . . 386
smtrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS_EW > . . . 387
smtrat::cadcells::operators::properties::cell_connected . . . 387
smtrat::cadcells::representation::approximation::CellApproximator . . . 388
smtrat::cadcells::datastructures::CellRepresentation< P > . . . 388
benchmax::slurm::ChunkedSubmitfileProperties . . . 389
Minisat::Clause . . . 391
smtrat::mcsat::ClauseChain . . . 396

smtrat::sat::detail::ClauseChecker . . . 398
smtrat::CMakeOptionPrinter . . . 399
Minisat::CMap< T > . . . 399
smtrat::CollectionWithOrigins< Element, Origin > . . . 454
smtrat::CollectionWithOrigins< carl::Variable, FormulaT > . . . 454
smtrat::ICPModule< Settings >::comp . . . 455
smtrat::cadcells::datastructures::CompoundMaxMin . . . 456
smtrat::cadcells::datastructures::CompoundMinMax . . . 456
smtrat::vs::Condition . . . 457
smtrat::mcsat::fm::ConflictGenerator< Comparator > . . . 462
smtrat::cad::ConflictGraph . . . 462
smtrat::parser::Theories::ConstantAdder . . . 464
smtrat::covering_ng::formula::node_ds::CONSTRAINT . . . 465
smtrat::cad::CADConstraints< BT >::ConstraintComparator . . . 465
smtrat::cad::preprocessor::ConstraintUpdate . . . 465
smtrat::ConstrTree . . . 466
smtrat::LRAModule< Settings >::Context . . . 466
smtrat::icp::ContractionCandidate . . . 467
smtrat::icp::contractionCandidateComp . . . 472
smtrat::icp::ContractionCandidateManager . . . 472
smtrat::parser::conversion::Converter< To > . . . 474
smtrat::parser::conversion::Converter< FormulaT > . . . 474
smtrat::parser::conversion::Converter< Poly > . . . 475
smtrat::parser::conversion::Converter< Res > . . . 474
smtrat::parser::conversion::Converter< types::BVTerm > . . . 475
benchmax::settings::CoreSettings . . . 476
smtrat::settings::CoreSettings . . . 477
smtrat::parser::types::CoreTheory . . . 480
smtrat::cadcells::representation::covering< H > . . . 481
smtrat::mcsat::arithmetic::Covering . . . 481
smtrat::cadcells::representation::covering< CoveringHeuristic::BIGGEST_CELL_COVERING > . . . 483
smtrat::cadcells::representation::covering< CoveringHeuristic::BIGGEST_CELL_COVERING_EW > . . . 483
smtrat::cadcells::representation::covering< CoveringHeuristic::BIGGEST_CELL_COVERING_MIN_↩
TDEG > . . . 483
smtrat::cadcells::representation::covering< CoveringHeuristic::CHAIN_COVERING > . . . 484
smtrat::cadcells::datastructures::CoveringDescription . . . 484
smtrat::cadcells::datastructures::CoveringRepresentation< P > . . . 511
smtrat::covering_ng::CoveringResult< op > . . . 512
smtrat::mcsat::onecellcad::recursive::CoverNullification . . . 514
smtrat::CycleEnumerator< FHG, Collector > . . . 595
benchmax::Database . . . 596
benchmax::DBAL . . . 598
Minisat::DeepEqual< K > . . . 601
Minisat::DeepHash< K > . . . 602
smtrat::mcsat::fm::DefaultComparator . . . 602
smtrat::mcsat::fm::DefaultSettings . . . 603
smtrat::mcsat::smtaf::DefaultSettings . . . 603
smtrat::mcsat::vs::DefaultSettings . . . 603
smtrat::cad::projection_compare::degree . . . 604
smtrat::mcsat::onecellcad::recursive::DegreeAscending . . . 604
smtrat::analyzer::DegreeCollector . . . 604
smtrat::mcsat::onecellcad::recursive::DegreeDescending . . . 605
smtrat::cadcells::datastructures::DelineatedDerivation< Properties > . . . 605
smtrat::cadcells::datastructures::Delineation . . . 607
smtrat::cadcells::datastructures::DelineationInterval . . . 609
smtrat::mcsat::icp::Dependencies . . . 610
smtrat::cadcells::datastructures::DerivationRef< Properties > . . . 611
smtrat::mcsat::onecellcad::recursive::DontCoverNullification . . . 613

smtrat::cad::debug::DotSubgraph . . . 614
Minisat::DoubleRange . . . 616
smtrat::DynamicPriorityQueue< T, Compare > . . . 618
smtrat::cad::ProjectionGlobalInformation::ECData . . . 620
smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Edge . . . 621
smtrat::ElementWithOrigins< Element, Origin > . . . 622
smtrat::subtropical::Encoding . . . 649
smtrat::EQGraph< VertexName > . . . 650
smtrat::EQGraph< carl::UVariable > . . . 650
Minisat::Equal< K > . . . 653
Minisat::Equal< CRef > . . . 653
smtrat::parser::ErrorHandler . . . 653
smtrat::execution::ExecutionState . . . 681
smtrat::covering_ng::formula::ExhaustiveImplicants . . . 690
smtrat::mcsat::fm::Explanation< Settings > . . . 692
smtrat::mcsat::icp::Explanation . . . 692
smtrat::mcsat::nlsat::Explanation . . . 692
smtrat::mcsat::onecell::Explanation . . . 693
smtrat::mcsat::onecellcad::levelwise::Explanation< Setting1, Setting2 > . . . 694
smtrat::mcsat::onecellcad::recursive::Explanation< Setting1, Setting2 > . . . 694
smtrat::mcsat::vs::Explanation . . . 694
smtrat::mcsat::nlsat::ExplanationGenerator . . . 695
smtrat::mcsat::vs::ExplanationGenerator< Settings > . . . 695
smtrat::expression::Expression . . . 696
smtrat::expression::ExpressionContent . . . 699
smtrat::parser::ParserState::ExpressionScope . . . 704
smtrat::covering_ng::formula::node_ds::FALSE . . . 706
std::false_type
  smtrat::is_variant< T > . . . 1012
smtrat::mcsat::FastParallelExplanation< Backends > . . . 706
smtrat::mcsat::variableordering::detail::FeatureCollector< Objects > . . . 706
smtrat::parser::FixedWidthConstant< T > . . . 708
smtrat::ICPModule< Settings >::formulaPtrComp . . . 714
smtrat::FormulaWithOrigins . . . 715
smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices > . . . 717
smtrat::qe::fm::FourierMotzkinQE . . . 748
smtrat::mcsat::FullParallelExplanation< Backends > . . . 777
smtrat::cad::FullSampleComparator< Iterator, Strategy > . . . 777
smtrat::parser::FunctionInstantiator . . . 777
  smtrat::parser::UserFunctionInstantiator . . . 1833
smtrat::GBModuleState< Settings > . . . 809
boost::spirit::qi::grammar
  smtrat::parser::ScriptParser< smtrat::parser::SMTLIBParser > . . . 1567
  smtrat::parser::AttributeParser . . . 250
  smtrat::parser::AttributeValueParser . . . 251
  smtrat::parser::BinaryParser . . . 309
  smtrat::parser::HexadecimalParser . . . 844
  smtrat::parser::IdentifierParser . . . 908
  smtrat::parser::KeywordParser . . . 1015
  smtrat::parser::QEParser . . . 1500
  smtrat::parser::QualifiedIdentifierParser . . . 1501
  smtrat::parser::SExpressionParser . . . 1576
  smtrat::parser::ScriptParser< Callee > . . . 1567
  smtrat::parser::SimpleSymbolParser . . . 1580
  smtrat::parser::Skipper . . . 1583
  smtrat::parser::SortParser . . . 1600
  smtrat::parser::SortedVariableParser . . . 1599
  smtrat::parser::SpecConstantParser . . . 1600

smtrat::parser::StringParser . . . 1661
smtrat::parser::SymbolParser . . . 1695
smtrat::parser::TermParser . . . 1745
Minisat::Hash< K > . . . 838
std::hash< const smtrat::expression::ExpressionContent * > . . . 839
std::hash< smtrat::expression::BinaryExpression > . . . 839
std::hash< smtrat::expression::Expression > . . . 839
std::hash< smtrat::expression::ITEExpression > . . . 839
std::hash< smtrat::expression::NaryExpression > . . . 840
std::hash< smtrat::expression::QuantifierExpression > . . . 840
std::hash< smtrat::expression::UnaryExpression > . . . 840
std::hash< smtrat::lra::Variable< T1, T2 > > . . . 840
std::hash< smtrat::vs::Substitution > . . . 841
std::hash< std::vector< T > > . . . 841
std::hash_combiner . . . 842
Minisat::Heap< Comp > . . . 842
Minisat::Heap< VarOrderLt > . . . 842
smtrat::icp::HistoryNode . . . 845
smtrat::icp::IcpVariable . . . 903
smtrat::icp::icpVariableComp . . . 906
smtrat::parser::Identifier . . . 906
smtrat::cad::debug::IDSanitizer . . . 908
smtrat::covering_ng::formula::node_ds::IFF . . . 909
smtrat::mcsat::fm::IgnoreCoreSettings . . . 909
Implementation
smtrat::UnionFindInterface< T, Implementation > . . . 1802
smtrat::parser::IndexedFunctionInstantiator . . . 935
smtrat::cadcells::datastructures::IndexedRoot . . . 936
smtrat::cadcells::datastructures::IndexedRootOrdering . . . 937
smtrat::cadcells::datastructures::IndexedRootRelation . . . 938
smtrat::InequalitiesTable< Settings > . . . 939
smtrat::lra::Bound< T1, T2 >::Info . . . 942
smtrat::mcsat::InformationGetter . . . 943
smtrat::parser::InstructionHandler . . . 947
smtrat::Executor< Strategy > . . . 684
smtrat::parseformula::FormulaCollector . . . 708
Minisat::Int64Range . . . 952
smtrat::cadcells::representation::util::IntervalCompare< T > . . . 1008
smtrat::covering_ng::IntervalCompare< op > . . . 1008
smtrat::mcsat::icp::IntervalPropagation . . . 1009
Minisat::IntRange . . . 1011
smtrat::is_sample_outside< S > . . . 1011
smtrat::is_sample_outside< IsSampleOutsideAlgorithm::DEFAULT > . . . 1012
smtrat::expression::ITEExpression . . . 1012
std::iterator
benchmax::RandomizationAdaptor< T >::iterator . . . 1013
smtrat::ModuleInput::IteratorCompare . . . 1014
benchmax::Jobs . . . 1014
Minisat::lbool . . . 1016
smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound . . . 1018
smtrat::Module::Lemma . . . 1019
Minisat::LessThan_default< T > . . . 1020
smtrat::cad::projection_compare::level . . . 1020
smtrat::cad::sample_compare::level . . . 1020
smtrat::cad::ProjectionLevelInformation::LevelInfo . . . 1021
smtrat::LevelWiseInformation< Settings > . . . 1024
smtrat::cad::LiftingTree< Settings > . . . 1027
smtrat::ICPModule< Settings >::linearVariable . . . 1031

smtrat::mcsat::ClauseChain::Link . . . 1031
std::list< T >
  smtrat::ModuleInput . . . 1194
Minisat::Lit . . . 1032
smtrat::Manager . . . 1095
Minisat::Map< K, D, H, E > . . . 1104
std::map< K, T >
  smtrat::VariantMap< std::string, Value > . . . 1868
  smtrat::VariantMap< Key, Value > . . . 1868
Minisat::Map< CRef, T, CRefHash > . . . 1104
smtrat::mcsat::fm::MaxSizeComparator . . . 1108
smtrat::MaxSMT< Solver, Strategy > . . . 1109
smtrat::MaxSMT< Strategy, MaxSMTStrategy::LINEAR_SEARCH > . . . 1109
smtrat::maxsmt::MaxSMTBackend< Solver, Strategy > . . . 1110
smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::FU_MALIK_INCREMENTAL > . . . 1110
smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::LINEAR_SEARCH > . . . 1110
smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::MSU3 > . . . 1111
smtrat::mcsat::MCSATBackend< Settings > . . . 1137
smtrat::mcsat::MCSATBackend< typename Settings::MCSATSettings > . . . 1137
smtrat::mcsat::MCSATMixin< Settings > . . . 1139
smtrat::mcsat::MCSATMixin< typename Settings::MCSATSettings > . . . 1139
smtrat::covering_ng::formula::Minimal . . . 1146
smtrat::mcsat::fm::MinSizeComparator . . . 1152
smtrat::mcsat::fm::MinVarCountComparator . . . 1153
smtrat::cad::MISGeneration< heuristic > . . . 1153
smtrat::Module . . . 1167
  smtrat::ICPModule< ICPSettings4 > . . . 876
  smtrat::ICPModule< ICPSettings1 > . . . 876
  smtrat::LRAModule< LRASettingsICP > . . . 1037
  smtrat::LRAModule< LRASettings1 > . . . 1037
  smtrat::BVModule< Settings > . . . 340
  smtrat::CSplitModule< Settings > . . . 514
  smtrat::CoCoAGBModule< Settings > . . . 427
  smtrat::CoveringNGModule< Settings > . . . 485
  smtrat::CubeLIAModule< Settings > . . . 540
  smtrat::CurryModule< Settings > . . . 567
  smtrat::FouMoModule< Settings > . . . 720
  smtrat::GBModule< Settings > . . . 778
  smtrat::ICPModule< Settings > . . . 876
  smtrat::IncWidthModule< Settings > . . . 909
  smtrat::IntBlastModule< Settings > . . . 953
  smtrat::IntEqModule< Settings > . . . 980
  smtrat::LRAModule< Settings > . . . 1037
  smtrat::NRAILModule< Settings > . . . 1267
  smtrat::NewCADModule< Settings > . . . 1205
  smtrat::NewCoveringModule< Settings > . . . 1233
  smtrat::PBGaussModule< Settings > . . . 1316
  smtrat::PBPPModule< Settings > . . . 1345
  smtrat::PModule . . . 1402
    smtrat::BEModule< Settings > . . . 276
    smtrat::CNFerModule . . . 400
    smtrat::EMModule< Settings > . . . 623
    smtrat::ESModule< Settings > . . . 653
    smtrat::FPPModule< Settings > . . . 749
    smtrat::GBPPModule< Settings > . . . 810
    smtrat::ICEModule< Settings > . . . 849
    smtrat::LVEModule< Settings > . . . 1070
    smtrat::MCBModule< Settings > . . . 1111

smtrat::PFEModule< Settings > . . . 1375
smtrat::SplitSOSModule< Settings > . . . 1601
smtrat::SymmetryModule< Settings > . . . 1696
smtrat::SATModule< Settings > . . . 1535
smtrat::STropModule< Settings > . . . 1661
smtrat::UFCegarModule< Settings > . . . 1770
smtrat::UnionFindModule< Settings > . . . 1803
smtrat::VSModule< Settings > . . . 1913
smtrat::settings::ModuleSettings . . . 1200
smtrat::subtropical::Moment . . . 1202
smtrat::expression::NaryExpression . . . 1203
benchmax::ssh::Node . . . 1263
smtrat::covering_ng::formula::node_ds::Node . . . 1264
smtrat::covering_ng::formula::node_ds::NodeContent . . . 1265
smtrat::mcsat::onecellcad::recursive::NoHeuristic . . . 1266
smtrat::covering_ng::CoveringResult< op >::NONE . . . 1266
smtrat::covering_ng::formula::node_ds::NOT . . . 1266
smtrat::lra::Numeric . . . 1294
smtrat::execution::Objective . . . 1300
Minisat::OccLists< Idx, Vec, Deleted > . . . 1300
Minisat::OccLists< Minisat::Lit, Minisat::vec< Minisat::Watcher >, WatcherDeleted > . . . 1300
smtrat::mcsat::onecellcad::OneCellCAD . . . 1301
smtrat::mcsat::onecellcad::levelwise::LevelwiseCAD . . . 1022
smtrat::mcsat::onecellcad::recursive::RecursiveCAD . . . 1508
benchmax::settings::OperationSettings . . . 1303
smtrat::Optimization< Solver > . . . 1304
smtrat::Optimization< Strategy > . . . 1304
Minisat::Option . . . 1305
Minisat::BoolOption . . . 320
Minisat::DoubleOption . . . 614
Minisat::IntOption . . . 1009
Minisat::StringOption . . . 1660
Minisat::Option::OptionLt . . . 1307
smtrat::covering_ng::formula::node_ds::OR . . . 1307
smtrat::cad::Origin . . . 1307
smtrat::cad::preprocessor::Origins . . . 1310
Minisat::OutOfMemoryException . . . 1311
smtrat::parser::OutputWrapper . . . 1311
Minisat::Map< K, D, H, E >::Pair . . . 1311
smtrat::mcsat::ParallelExplanation< Backends > . . . 1312
smtrat::parser::ParserSettings . . . 1312
smtrat::parser::ParserState . . . 1312
smtrat::PersistentUnionFind . . . 1373
smtrat::cadcells::operators::properties::poly_additional_root_outside . . . 1428
smtrat::cadcells::operators::properties::poly_irreducible_semi_sgn_inv . . . 1429
smtrat::cadcells::operators::properties::poly_irreducible_sgn_inv . . . 1430
smtrat::cadcells::operators::properties::poly_ord_inv . . . 1430
smtrat::cadcells::operators::properties::poly_ord_inv_base . . . 1431
smtrat::cadcells::operators::properties::poly_pdel . . . 1432
smtrat::cadcells::operators::properties::poly_semi_sgn_inv . . . 1432
smtrat::cadcells::operators::properties::poly_sgn_inv . . . 1433
smtrat::cadcells::representation::util::PolyDelineation . . . 1434
smtrat::cadcells::representation::util::PolyDelineations . . . 1434
smtrat::cad::ProjectionPolynomialInformation::PolyInfo . . . 1434
smtrat::cad::PolynomialComparator< PolynomialGetter > . . . 1435
smtrat::cad::PolynomialLiftingQueue< PolynomialGetter > . . . 1435
smtrat::cad::PolynomialLiftingQueue< smtrat::cad::BaseProjection > . . . 1435
smtrat::cadcells::datastructures::PolyPool . . . 1437

smtrat::cadcells::datastructures::detail::PolyProperties . . . 1438
smtrat::cadcells::datastructures::PolyRef . . . 1439
smtrat::PolyTree . . . 1439
smtrat::PolyTreeContent . . . 1440
smtrat::cad::Preprocessor . . . 1442
smtrat::cad::PreprocessorSettings . . . 1443
benchmax::settings::PresetSettings . . . 1444
std::priority_queue< T >
  smtrat::PriorityQueue< Iterator, Comparator > . . . 1444
  smtrat::PriorityQueue< QueueEntry, smtrat::cad::PolynomialComparator > . . . 1444
  smtrat::PriorityQueue< QueueEntry, ProjectionCandidateComparator > . . . 1444
  smtrat::PriorityQueue< T, Compare > . . . 1444
smtrat::cad::projection_compare::ProjectionComparator< Strategy > . . . 1485
smtrat::cad::projection_compare::ProjectionComparator< Settings::projectionComparator > . . . 1485
smtrat::cad::projection_compare::ProjectionComparator_impl< Args > . . . 1487
smtrat::cad::projection_compare::ProjectionComparator_impl< degree, lt > . . . 1487
  smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::D > . . . 1485
smtrat::cad::projection_compare::ProjectionComparator_impl< level, gt, degree, lt > . . . 1487
  smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::LD > . . . 1486
smtrat::cad::projection_compare::ProjectionComparator_impl< level, lt, degree, lt > . . . 1487
  smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::lD > . . . 1486
smtrat::cad::projection_compare::ProjectionComparator_impl< type, gt, degree, lt > . . . 1487
  smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::PD > . . . 1486
smtrat::cad::projection_compare::ProjectionComparator_impl< type, lt, degree, lt > . . . 1487
  smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::SD > . . . 1486
smtrat::cad::ProjectionGlobalInformation . . . 1487
smtrat::cad::ProjectionInformation . . . 1488
smtrat::cad::ProjectionLevelInformation . . . 1490
smtrat::cad::ProjectionOperator . . . 1491
smtrat::cad::ProjectionPolynomialInformation . . . 1492
smtrat::cadcells::datastructures::Projections . . . 1493
smtrat::cadcells::operators::mccallum_filtered_impl::PropertiesSet . . . 1495
smtrat::cadcells::operators::PropertiesSet< Op > . . . 1496
smtrat::cadcells::operators::PropertiesSet< op::mccallum > . . . 1496
smtrat::cadcells::datastructures::PropertiesT< Ts > . . . 1496
smtrat::cadcells::datastructures::PropertiesT< Ts... > . . . 1496
  smtrat::cadcells::datastructures::PropertiesT< T, Ts... > . . . 1497
smtrat::cadcells::datastructures::PropertiesTContent< T, is_flag > . . . 1497
smtrat::cadcells::datastructures::PropertiesTContent< T, false > . . . 1497
smtrat::cadcells::datastructures::PropertiesTContent< T, T::is_flag > . . . 1497
smtrat::cadcells::datastructures::PropertiesTContent< T, true > . . . 1497
smtrat::cadcells::datastructures::property_hash< T > . . . 1497
smtrat::PseudoBoolEncoder . . . 1498
  smtrat::CardinalityEncoder . . . 383
  smtrat::ExactlyOneCommanderEncoder . . . 679
  smtrat::LongFormulaEncoder . . . 1036
  smtrat::MixedSignEncoder . . . 1154
  smtrat::RNSEncoder . . . 1517
  smtrat::ShortFormulaEncoder . . . 1577
  smtrat::TotalizerEncoder . . . 1767
smtrat::PseudoBoolNormalizer . . . 1499
smtrat::expression::QuantifierExpression . . . 1501
Minisat::Queue< T > . . . 1502
smtrat::mcsat::icp::QueueEntry . . . 1503
benchmax::RandomizationAdaptor< T > . . . 1504
smtrat::RationalCapsule . . . 1505

boost::spirit::qi::real_parser
smtrat::parser::DecimalParser . . . 601
boost::spirit::qi::real_policies
smtrat::parser::RationalPolicies . . . 1506
smtrat::mcsat::onecellcad::RealAlgebraicPoint< Number > . . . 1506
smtrat::mcsat::onecellcad::RealAlgebraicPoint< Rational > . . . 1506
Minisat::RegionAllocator< T > . . . 1513
Minisat::RegionAllocator< uint32_t > . . . 1513
Minisat::ClauseAllocator . . . 393
smtrat::parser::ErrorHandler::result< typename > . . . 1515
smtrat::cad::preprocessor::ResultantRule . . . 1515
benchmax::Results . . . 1516
smtrat::cadcells::operators::properties::root_ordering_holds . . . 1518
smtrat::cadcells::operators::properties::root_well_def . . . 1519
smtrat::cadcells::datastructures::RootFunction . . . 1519
smtrat::mcsat::arithmetic::RootIndexer< RANT > . . . 1521
smtrat::mcsat::arithmetic::RootIndexer< typename Poly::RootType > . . . 1521
smtrat::mcsat::arithmetic::RootIndexer< typename Polynomial::RootType > . . . 1521
smtrat::cad::Sample . . . 1522
smtrat::cad::sample_compare::SampleComparator< Iterator, Strategy > . . . 1524
smtrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::T > . . . 777
smtrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::Type > . . . 777
smtrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::Value > . . . 777
smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::T > . . . 1524
smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Type > . . . 1524
smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Type > . . . 1528
smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Value > . . . 1524
smtrat::cad::sample_compare::SampleComparator_impl< It, Args > . . . 1529
smtrat::cad::sample_compare::SampleComparator_impl< Iterator, level, gt, size, lt > . . . 1529
smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LS > . . . 1525
smtrat::cad::sample_compare::SampleComparator_impl< Iterator, level, gt, type, gt > . . . 1529
smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LT > . . . 1525
smtrat::cad::sample_compare::SampleComparator_impl< Iterator, level, gt, type, gt, absvalue, lt > . . . 1529
smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LTA > . . . 1525
smtrat::cad::sample_compare::SampleComparator_impl< Iterator, level, gt, type, gt, size, lt > . . . 1529
smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LTS > . . . 1525
smtrat::cad::sample_compare::SampleComparator_impl< Iterator, level, gt, type, gt, size, lt, absvalue, lt >1529
smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LTSA > . . . 1526
smtrat::cad::sample_compare::SampleComparator_impl< Iterator, size, lt > . . . 1529
smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::S > . . . 1526
smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Value > . . . 1528
smtrat::cad::sample_compare::SampleComparator_impl< Iterator, type, gt > . . . 1529
smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::T > . . . 1526
smtrat::cad::sample_compare::SampleComparator_impl< Iterator, type, gt, level, gt, size, lt, absvalue, lt >1529
smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::TLSA > . . . 1527
smtrat::cad::sample_compare::SampleComparator_impl< Iterator, type, gt, size, lt > . . . 1529
smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::TS > . . . 1527
smtrat::cad::sample_compare::SampleComparator_impl< Iterator, type, gt, size, lt, absvalue, lt > . . . 1529
smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::TSA > . . . 1527
smtrat::cadcells::datastructures::SampledDerivation< Properties > . . . 1529
smtrat::SampledDerivationRefCompare . . . 1532
smtrat::cad::SampleIteratorQueue< Iterator, Comparator > . . . 1532
smtrat::cad::SampleIteratorQueue< Iterator, smtrat::cad::FullSampleComparator > . . . 1532
smtrat::cad::SampleIteratorQueue< Iterator, smtrat::cad::sample_compare::SampleComparator > . . . 1532
smtrat::covering_ng::sampling< S > . . . 1533

smtrat::sampling< S > . . . 1534
smtrat::covering_ng::sampling< SamplingAlgorithm::LOWER_UPPER_BETWEEN_SAMPLING > . . . 1534
smtrat::sampling< SamplingAlgorithm::LOWER_UPPER_BETWEEN_SAMPLING > . . . 1534
smtrat::parser::ParserState::ScriptScope . . . 1569
smtrat::mcsat::onecellcad::Section . . . 1570
smtrat::onecellcad::recursive::Section . . . 1570
smtrat::mcsat::onecellcad::levelwise::SectionHeuristic1 . . . 1571
smtrat::mcsat::onecellcad::levelwise::SectionHeuristic2 . . . 1571
smtrat::mcsat::onecellcad::levelwise::SectionHeuristic3 . . . 1571
smtrat::mcsat::onecellcad::Sector . . . 1572
smtrat::onecellcad::recursive::Sector . . . 1572
smtrat::mcsat::onecellcad::levelwise::SectorHeuristic1 . . . 1573
smtrat::mcsat::onecellcad::levelwise::SectorHeuristic2 . . . 1573
smtrat::mcsat::onecellcad::levelwise::SectorHeuristic3 . . . 1574
smtrat::subtropical::Separator . . . 1574
smtrat::mcsat::SequentialAssignment< Backends > . . . 1575
smtrat::mcsat::SequentialExplanation< Backends > . . . 1575
carl::settings::Settings
  benchmax::settings::Settings . . . 1575
  smtrat::settings::Settings . . . 1575
carl::settings::SettingsParser
  benchmax::SettingsParser . . . 1575
  smtrat::SettingsParser . . . 1575
smtrat::CNFerModule::SettingsType . . . 1576
benchmax::simple_parser . . . 1578
smtrat::parser::Theories::SimpleSortAdder . . . 1579
smtrat::expression::simplifier::Simplifier . . . 1581
smtrat::expression::simplifier::SimplifierChainCaller< chainID > . . . 1581
smtrat::expression::simplifier::SimplifierChainCaller< 0 > . . . 1581
carl::Singleton
  benchmax::SettingsParser . . . 1575
  benchmax::settings::Settings . . . 1575
  smtrat::LOG . . . 1035
  smtrat::MonomialMappingByVariablePool . . . 1202
  smtrat::PolyTreePool . . . 1441
  smtrat::SettingsComponents . . . 1575
  smtrat::SettingsParser . . . 1575
  smtrat::cadcells::representation::approximation::ApxCriteria . . . 237
  smtrat::expression::ExpressionPool . . . 702
  smtrat::resource::Limiter . . . 1030
  smtrat::settings::Settings . . . 1575
  smtrat::validation::ValidationCollector . . . 1834
smtrat::cad::sample_compare::size . . . 1583
benchmax::settings::SlurmBackendSettings . . . 1586
smtrat::parser::SMTLIBParser . . . 1587
smtrat::execution::SoftAssertion . . . 1598
smtrat::settings::SolverSettings . . . 1598
benchmax::settings::SSHBackendSettings . . . 1630
benchmax::ssh::SSHConnection . . . 1631
smtrat::vs::State . . . 1633
boost::static_visitor
  smtrat::parser::conversion::VariantConverter< types::BVTerm > . . . 1866
  smtrat::parser::conversion::VariantVariantConverter< types::AttributeValue > . . . 1870
  smtrat::parser::conversion::VariantVariantConverter< types::TermType > . . . 1870
  smtrat::expression::ExpressionConverter . . . 700
  smtrat::expression::ExpressionModifier . . . 701
  smtrat::expression::ExpressionTypeChecker< T > . . . 704
  smtrat::expression::ExpressionVisitor . . . 705

smtrat::expression::simplifier::BaseSimplifier . . . 269
smtrat::expression::simplifier::DuplicateSimplifier . . . 617
smtrat::expression::simplifier::MergeSimplifier . . . 1145
smtrat::expression::simplifier::NegationSimplifier . . . 1204
smtrat::expression::simplifier::SingletonSimplifier . . . 1582
smtrat::parser::Instantiator< V, T > . . . 945
smtrat::parser::conversion::VariantConverter< Res > . . . 1866
smtrat::parser::conversion::VariantVariantConverter< Res > . . . 1870
smtrat::StaticUnionFind . . . 1657
Statistics
smtrat::Module::ModuleStatistics . . . 1201
smtrat::analyzer::AnalyzerStatistics . . . 236
smtrat::statistics::StatisticsSettings . . . 1658
smtrat::StrategyGraph . . . 1658
benchmax::slurm::SubmitfileProperties . . . 1688
smtrat::vs::Substitution . . . 1689
smtrat::vs::substitutionPointerEqual . . . 1693
smtrat::vs::substitutionPointerHash . . . 1693
smtrat::cadcells::datastructures::SymbolicInterval . . . 1693
boost::spirit::qi::symbols
smtrat::parser::LogicParser . . . 1036
smtrat::parser::QuantifierParser . . . 1502
smtrat::lra::Tableau< Settings, T1, T2 > . . . 1722
smtrat::lra::Tableau< typename Settings::Tableau_settings, LRABoundType, LRAEntryType > . . . 1722
smtrat::lra::TableauEntry< T1, T2 > . . . 1740
smtrat::lra::TableauEntry< LRABoundType, LRAEntryType > . . . 1740
smtrat::cadcells::datastructures::TaggedIndexedRoot . . . 1743
smtrat::mcsat::onecellcad::TagPoly . . . 1744
smtrat::Task . . . 1744
smtrat::mcsat::vs::helper::TestCandidate . . . 1747
smtrat::parser::Theories . . . 1747
smtrat::parser::TheoryError . . . 1750
smtrat::mcsat::TheoryLevel . . . 1751
smtrat::Module::TheoryPropagation . . . 1752
smtrat::ThreadPool . . . 1757
smtrat::cad::debug::TikzBasePrinter . . . 1757
smtrat::cad::debug::TikzDAGPrinter . . . 1758
smtrat::cad::debug::TikzTreePrinter . . . 1761
smtrat::cad::debug::TikzHistoryPrinter . . . 1760
benchmax::Tool . . . 1762
benchmax::MathSAT . . . 1106
benchmax::Minisat . . . 1147
benchmax::Minisatp . . . 1150
benchmax::SMTRAT . . . 1590
benchmax::SMTRAT_OPB . . . 1595
benchmax::SMTRAT_Analyzer . . . 1593
benchmax::Z3 . . . 1944
benchmax::settings::ToolSettings . . . 1766
smtrat::TotalizerTree . . . 1769
smtrat::covering_ng::formula::node_ds::TRUE . . . 1769
smtrat::cad::projection_compare::type . . . 1769
smtrat::cad::sample_compare::type . . . 1769
boost::spirit::qi::uint_parser
smtrat::parser::NumeralParser . . . 1293
smtrat::expression::UnaryExpression . . . 1797
smtrat::cad::debug::UnifiedData . . . 1797
smtrat::parser::types::UninterpretedTheory . . . 1798
smtrat::SATModule< Settings >::UnorderedClauseLookup::UnorderedClauseHasher . . . 1830

smtrat::UnsatCore< Solver, Strategy > . . . 1830
smtrat::UnsatCore< Strategy, UnsatCoreStrategy::ModelExclusion > . . . 1830
smtrat::unsatcore::UnsatCoreBackend< Solver, Strategy > . . . 1831
smtrat::unsatcore::UnsatCoreBackend< Solver, UnsatCoreStrategy::ModelExclusion > . . . 1831
smtrat::vs::unsignedTripleCmp . . . 1832
smtrat::validation::ValidationPoint . . . 1834
smtrat::validation::ValidationPrinter< SOF > . . . 1835
smtrat::validation::ValidationSettings . . . 1835
smtrat::lra::Value< T > . . . 1836
smtrat::lra::Value< LRABoundType > . . . 1836
smtrat::lra::Value< T1 > . . . 1836
smtrat::lra::Variable< T1, T2 > . . . 1843
smtrat::vb::Variable< T > . . . 1854
smtrat::lra::Variable< LRABoundType, LRAEntryType > . . . 1843
smtrat::vb::VariableBounds< T > . . . 1857
smtrat::vb::VariableBounds< ConstraintT > . . . 1857
smtrat::vb::VariableBounds< FormulaT > . . . 1857
smtrat::VariableCapsule . . . 1864
smtrat::mcsat::variableordering::VariableIDs . . . 1865
smtrat::VariableRewriteRule . . . 1865
smtrat::VarSchedulerBase . . . 1871
- smtrat::VarSchedulerMcsatBase . . . 1880
  - smtrat::TheoryVarSchedulerStatic< vot > . . . 1753
  - smtrat::VarSchedulerMcsatActivityPreferTheory< vot > . . . 1877
  - smtrat::VarSchedulerMcsatBooleanFirst< vot > . . . 1884
  - smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler > . . . 1887
  - smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities > . . . 1891
  - smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot > . . . 1895
- smtrat::VarSchedulerMinisat . . . 1898
  - smtrat::VarSchedulerFixedRandom . . . 1874
- smtrat::VarSchedulerRandom . . . 1902
- smtrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic > . . . 1905

Minisat::vec< T > . . . 1908
Minisat::vec< bool > . . . 1908
Minisat::vec< char > . . . 1908
Minisat::vec< double > . . . 1908
Minisat::vec< Idx > . . . 1908
Minisat::vec< int > . . . 1908
Minisat::vec< Minisat::CRef > . . . 1908
Minisat::vec< Minisat::lbool > . . . 1908
Minisat::vec< Minisat::Lit > . . . 1908
Minisat::vec< Minisat::Map::Pair > . . . 1908
Minisat::vec< Minisat::vec< Minisat::Lit > > . . . 1908
Minisat::vec< std::pair< Abstraction *, Abstraction * > > . . . 1908
Minisat::vec< unsigned > . . . 1908
Minisat::vec< VarData > . . . 1908
Minisat::vec< Vec > . . . 1908
std::vector< T >
- smtrat::cad::projection::Reducta< Poly > . . . 1512
- smtrat::parser::SExpressionSequence< T > . . . 1576

smtrat::parser::conversion::VectorVariantConverter< Res > . . . 1911
smtrat::parser::conversion::VectorVariantConverter< types::BVTerm > . . . 1911
smtrat::subtropical::Vertex . . . 1912
Minisat::Watcher . . . 1942
smtrat::ICPModule< Settings >::weights . . . 1943
benchmax::XMLWriter . . . 1944
smtrat::covering_ng::formula::node_ds::XOR . . . 1944

0.13 Data Structure Index

0.13.1 Data Structures

Here are the data structures with brief descriptions:

smtrat::AbstractModuleFactory . . . 232
smtrat::parser::AbstractTheory
Base class for all theories . . . 232
smtrat::cad::sample_compare::absvalue . . . 235
smtrat::analyzer::AnalysisSettings . . . 235
smtrat::analyzer::AnalyzerStatistics . . . 236
smtrat::covering_ng::formula::node_ds::AND . . . 236
smtrat::AnnotatedBVTerm . . . 236
smtrat::cadcells::representation::approximation::ApxCriteria . . . 237
smtrat::cadcells::representation::approximation::ApxSettings . . . 238
benchmax::slurm::ArchiveProperties
All properties needed to archive log files . . . 240
smtrat::parser::ArithmeticTheory
Implements the theory of arithmetic, including LRA, LIA, NRA and NIA . . . 241
smtrat::parser::types::ArithmeticTheory
Types of the arithmetic theory . . . 244
smtrat::execution::Assertion . . . 244
smtrat::cad::preprocessor::AssignmentCollector . . . 244
smtrat::mcsat::arithmetic::AssignmentFinder . . . 246
smtrat::mcsat::smtaf::AssignmentFinder< Settings > . . . 246
smtrat::mcsat::arithmetic::AssignmentFinder_ctx . . . 246
smtrat::mcsat::arithmetic::AssignmentFinder_detail . . . 247
smtrat::mcsat::smtaf::AssignmentFinder_SMT . . . 248
smtrat::cadcells::datastructures::detail::AssignmentProperties . . . 249
smtrat::parser::Attribute
Represents an Attribute . . . 249
smtrat::parser::AttributeParser . . . 250
smtrat::parser::AttributeValueParser . . . 251
smtrat::AxiomFactory . . . 252
benchmax::Backend
Base class for all backends . . . 253
smtrat::Backend< Settings > . . . 255
smtrat::BackendLink . . . 257
smtrat::BackendSynchronisation . . . 258
smtrat::Backtrackable< UnionFind > . . . 259
smtrat::mcsat::onecell::Base . . . 260
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 . . . 261
smtrat::cad::BaseProjection< Settings > . . . 263
smtrat::cad::BaseSettings . . . 268
smtrat::expression::simplifier::BaseSimplifier . . . 269
smtrat::cad::Origin::BaseType . . . 271
smtrat::mcsat::onecell::BCApproximationSettings . . . 273
smtrat::mcsat::onecell::BCFilteredAllSelectiveSettings . . . 273
smtrat::mcsat::onecell::BCFilteredAllSettings . . . 274
smtrat::mcsat::onecell::BCFilteredBoundsSettings . . . 274
smtrat::mcsat::onecell::BCFilteredSamplesSettings . . . 275
smtrat::mcsat::onecell::BCFilteredSettings . . . 275
smtrat::mcsat::onecell::BCSettings . . . 276
smtrat::BEModule< Settings > . . . 276
benchmax::BenchmarkResult
Results for a single benchmark run . . . 302

benchmax::BenchmarkSet
A set of benchmarks from some common base directory . . . 304
benchmax::settings::BenchmarkSettings
Settings for benchmarks . . . 305
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 . . . 306
smtrat::expression::BinaryExpression . . . 308
smtrat::parser::BinaryParser
Parses binaries: `#b[01]+` . . . 309
smtrat::parser::BitvectorTheory
Implements the theory of bitvectors . . . 310
smtrat::parser::types::BitvectorTheory
Types of the theory of bitvectors . . . 312
smtrat::BlastedConstr . . . 313
smtrat::BlastedPoly . . . 314
smtrat::mcsat::Bookkeeping
Represent the trail, i.e . . . 315
smtrat::covering_ng::formula::node_ds::BOOL . . . 317
smtrat::parser::BooleanEncodingTheory
Implements the theory of bitvectors . . . 317
Minisat::BoolOption . . . 320
smtrat::cadcells::datastructures::Bound
Bound type of a SymbolicInterval . . . 321
smtrat::lra::Bound< T1, T2 >
Stores a bound, which could be an upper "<= b" or a lower bound ">= b" for a bound value b . 322
smtrat::mcsat::fm::Bound . . . 331
smtrat::vb::Bound< T >
Class for the bound of a variable . . . 332
smtrat::Branching
Stores all necessary information of an branch, which can be used to detect probable infinite loop of branchings . . . 337
smtrat::BVAnnotation . . . 338
smtrat::BVDirectEncoder . . . 340
smtrat::BVModule< Settings > . . . 340
smtrat::cad::CAD< Settings > . . . 368
smtrat::qe::cad::CAD< Settings > . . . 371
smtrat::cad::CADConstraints< BT > . . . 373
smtrat::cad::CADCore< CH > . . . 377
smtrat::cad::CADCore< CoreHeuristic::BySample > . . . 377
smtrat::cad::CADCore< CoreHeuristic::EnumerateAll > . . . 378
smtrat::cad::CADCore< CoreHeuristic::Interleave > . . . 378
smtrat::cad::CADCore< CoreHeuristic::PreferProjection > . . . 379
smtrat::cad::CADCore< CoreHeuristic::PreferSampling > . . . 379
smtrat::qe::cad::CADElimination . . . 379
smtrat::cad::CADPreprocessor . . . 380
smtrat::cad::CADPreprocessorSettings . . . 381
smtrat::qe::cad::CADSettings . . . 382
smtrat::CardinalityEncoder . . . 383
smtrat::cadcells::representation::cell< H >
Note: If connected(i) holds, then the indexed root ordering must contain an ordering between the interval bounds . . . 384
smtrat::cadcells::representation::cell< CellHeuristic::BIGGEST_CELL > . . . 385
smtrat::cadcells::representation::cell< CellHeuristic::BIGGEST_CELL_APPROXIMATION > . . . 385
smtrat::cadcells::representation::cell< CellHeuristic::BIGGEST_CELL_EW > . . . 385
smtrat::cadcells::representation::cell< CellHeuristic::CHAIN_EQ > . . . 386
smtrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS > . . . 386
smtrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS_EQ > . . . 386

smtrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS_EW > 387
smtrat::cadcells::operators::properties::cell_connected 387
smtrat::cadcells::representation::approximation::CellApproximator 388
smtrat::cadcells::datastructures::CellRepresentation< P >
Represents a cell 388
benchmax::slurm::ChunkedSubmitfileProperties
All properties needed to create a submit file 389
Minisat::Clause 391
Minisat::ClauseAllocator 393
smtrat::mcsat::ClauseChain
An explanation is either a single clause or a chain of clauses, satisfying the following properties: 396
smtrat::sat::detail::ClauseChecker 398
smtrat::CMakeOptionPrinter 399
Minisat::CMap< T > 399
smtrat::CNFerModule 400
smtrat::CoCoAGBModule< Settings > 427
smtrat::CollectionWithOrigins< Element, Origin > 454
smtrat::ICPModule< Settings >::comp
Typedefs: 455
smtrat::cadcells::datastructures::CompoundMaxMin
Represents the maximum function of the contained indexed root functions 456
smtrat::cadcells::datastructures::CompoundMinMax
Represents the minimum function of the contained indexed root functions 456
smtrat::vs::Condition 457
benchmax::CondorBackend
Backend for the HTCondor batch system 459
smtrat::mcsat::fm::ConflictGenerator< Comparator > 462
smtrat::cad::ConflictGraph
Representation of a bipartite graph (C+S, E) of vertices C and S, representing the constraints and samples, respectively 462
smtrat::parser::Theories::ConstantAdder
Helper functor for addConstants() method 464
smtrat::covering_ng::formula::node_ds::CONSTRAINT 465
smtrat::cad::CADConstraints< BT >::ConstraintComparator 465
smtrat::cad::preprocessor::ConstraintUpdate 465
smtrat::ConstrTree 466
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 466
smtrat::icp::ContractionCandidate 467
smtrat::icp::contractionCandidateComp 472
smtrat::icp::ContractionCandidateManager 472
smtrat::parser::conversion::Converter< To > 474
smtrat::parser::conversion::Converter< FormulaT > 474
smtrat::parser::conversion::Converter< Poly > 475
smtrat::parser::conversion::Converter< types::BVTerm > 475
benchmax::settings::CoreSettings
Core settings 476
smtrat::settings::CoreSettings 477
smtrat::parser::CoreTheory
Implements the core theory of the booleans 478
smtrat::parser::types::CoreTheory
Types of the core theory 480
smtrat::cadcells::representation::covering< H > 481
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 481
smtrat::cadcells::representation::covering< CoveringHeuristic::BIGGEST_CELL_COVERING > 483

smtrat::cadcells::representation::covering< CoveringHeuristic::BIGGEST_CELL_COVERING_EW > . . 483
smtrat::cadcells::representation::covering< CoveringHeuristic::BIGGEST_CELL_COVERING_MIN_TDEG > 483
smtrat::cadcells::representation::covering< CoveringHeuristic::CHAIN_COVERING > 484
smtrat::cadcells::datastructures::CoveringDescription
Describes a covering of the real line by SymbolicIntervals (given an appropriate sample) . . . 484
smtrat::CoveringNGModule< Settings > 485
smtrat::cadcells::datastructures::CoveringRepresentation< P >
Represents a covering over a cell 511
smtrat::covering_ng::CoveringResult< op > 512
smtrat::mcsat::onecellcad::recursive::CoverNullification 514
smtrat::CSplitModule< Settings > 514
smtrat::CubeLIAModule< Settings > 540
smtrat::CurryModule< Settings > 567
smtrat::CycleEnumerator< FHG, Collector >
This class encapsulates an algorithm for enumerating all cycles 595
benchmax::Database
Dummy database that effectively disables storing to database. Set BENCHMAX_DATABASE to actually use a database 596
benchmax::DBAL 598
smtrat::parser::DecimalParser
Parses decimals: `numeral.0*numeral` 601
Minisat::DeepEqual< K > 601
Minisat::DeepHash< K > 602
smtrat::mcsat::fm::DefaultComparator
Does not order anything 602
smtrat::mcsat::fm::DefaultSettings 603
smtrat::mcsat::smtaf::DefaultSettings 603
smtrat::mcsat::vs::DefaultSettings 603
smtrat::cad::projection_compare::degree 604
smtrat::mcsat::onecellcad::recursive::DegreeAscending 604
smtrat::analyzer::DegreeCollector 604
smtrat::mcsat::onecellcad::recursive::DegreeDescending 605
smtrat::cadcells::datastructures::DelineatedDerivation< Properties >
A DelineatedDerivation is a BaseDerivation with a Delineation and an underlying SampledDerivation 605
smtrat::cadcells::datastructures::Delineation
Represents the delineation of a set of polynomials (at a sample), that is 607
smtrat::cadcells::datastructures::DelineationInterval
An interval of a delineation 609
smtrat::mcsat::icp::Dependencies 610
smtrat::cadcells::datastructures::DerivationRef< Properties >
A reference to a derivation, which is either 611
smtrat::mcsat::onecellcad::recursive::DontCoverNullification 613
smtrat::cad::debug::DotSubgraph 614
Minisat::DoubleOption 614
Minisat::DoubleRange 616
smtrat::expression::simplifier::DuplicateSimplifier 617
smtrat::DynamicPriorityQueue< T, Compare > 618
smtrat::cad::ProjectionGlobalInformation::ECData 620
smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Edge
Internal type of an edge 621
smtrat::ElementWithOrigins< Element, Origin > 622
smtrat::EMModule< Settings > 623
smtrat::subtropical::Encoding 649
smtrat::EQGraph< VertexName > 650
Minisat::Equal< K > 653
smtrat::parser::ErrorHandler 653

smtrat::ESModule< Settings > . . . 653
smtrat::ExactlyOneCommanderEncoder . . . 679
smtrat::execution::ExecutionState . . . 681
smtrat::Executor< Strategy > . . . 684
smtrat::covering_ng::formula::ExhaustiveImplicants . . . 690
smtrat::mcsat::fm::Explanation< Settings > . . . 692
smtrat::mcsat::icp::Explanation . . . 692
smtrat::mcsat::nlsat::Explanation . . . 692
smtrat::mcsat::onecell::Explanation . . . 693
smtrat::mcsat::onecellcad::levelwise::Explanation< Setting1, Setting2 > . . . 694
smtrat::mcsat::onecellcad::recursive::Explanation< Setting1, Setting2 > . . . 694
smtrat::mcsat::vs::Explanation . . . 694
smtrat::mcsat::nlsat::ExplanationGenerator . . . 695
smtrat::mcsat::vs::ExplanationGenerator< Settings > . . . 695
smtrat::expression::Expression . . . 696
smtrat::expression::ExpressionContent . . . 699
smtrat::expression::ExpressionConverter . . . 700
smtrat::expression::ExpressionModifier . . . 701
smtrat::expression::ExpressionPool . . . 702
smtrat::parser::ParserState::ExpressionScope . . . 704
smtrat::expression::ExpressionTypeChecker< T > . . . 704
smtrat::expression::ExpressionVisitor . . . 705
smtrat::covering_ng::formula::node_ds::FALSE . . . 706
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 706
smtrat::mcsat::variableordering::detail::FeatureCollector< Objects >
This class manages features that are used to valuate variables on objects . . . 706
smtrat::parser::FixedWidthConstant< T >
Represents a constant of a fixed width . . . 708
smtrat::parseformula::FormulaCollector . . . 708
smtrat::ICPModule< Settings >::formulaPtrComp . . . 714
smtrat::FormulaWithOrigins
Stores a formula along with its origins . . . 715
smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >
This class implements a forward hypergraph . . . 717
smtrat::FouMoModule< Settings >
A module which applies the Fourier-Motzkin algorithm . . . 720
smtrat::qe::fm::FourierMotzkinQE
Provides a simple implementation for Fourier Motzkin variable elimination for linear, existentially quantified constraints . . . 748
smtrat::FPPModule< Settings > . . . 749
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 . . . 777
smtrat::cad::FullSampleComparator< Iterator, Strategy > . . . 777
smtrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::T > . . . 777
smtrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::Type > . . . 777
smtrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::Value > . . . 777
smtrat::parser::FunctionInstantiator . . . 777
smtrat::GBModule< Settings >
A solver module based on Groebner basis . . . 778
smtrat::GBModuleState< Settings >
A class to save the current state of a GBModule . . . 809
smtrat::GBPPModule< Settings > . . . 810
Minisat::Hash< K > . . . 838
std::hash< const smtrat::expression::ExpressionContent * > . . . 839
std::hash< smtrat::expression::BinaryExpression > . . . 839

std::hash< smtrat::expression::Expression > . . . 839
std::hash< smtrat::expression::ITEExpression > . . . 839
std::hash< smtrat::expression::NaryExpression > . . . 840
std::hash< smtrat::expression::QuantifierExpression > . . . 840
std::hash< smtrat::expression::UnaryExpression > . . . 840
std::hash< smtrat::lra::Variable< T1, T2 > >
Implements std::hash for sort value . . . 840
std::hash< smtrat::vs::Substitution > . . . 841
std::hash< std::vector< T > > . . . 841
std::hash_combiner . . . 842
Minisat::Heap< Comp > . . . 842
smtrat::parser::HexadecimalParser
Parses hexadecimals: `#x[0-9a-fA-F]+` . . . 844
smtrat::icp::HistoryNode . . . 845
smtrat::ICEModule< Settings > . . . 849
smtrat::ICPModule< Settings > . . . 876
smtrat::icp::IcpVariable . . . 903
smtrat::icp::icpVariableComp . . . 906
smtrat::parser::Identifier . . . 906
smtrat::parser::IdentifierParser . . . 908
smtrat::cad::debug::IDSanitizer . . . 908
smtrat::covering_ng::formula::node_ds::IFF . . . 909
smtrat::mcsat::fm::IgnoreCoreSettings . . . 909
smtrat::IncWidthModule< Settings > . . . 909
smtrat::parser::IndexedFunctionInstantiator . . . 935
smtrat::cadcells::datastructures::IndexedRoot
Represents the i-th root of a multivariate polynomial at its main variable (given an appropriate sample) . . . 936
smtrat::cadcells::datastructures::IndexedRootOrdering
Describes an ordering of IndexedRoots . . . 937
smtrat::cadcells::datastructures::IndexedRootRelation
A relation between two roots . . . 938
smtrat::InequalitiesTable< Settings >
Datastructure for the GBModule . . . 939
smtrat::lra::Bound< T1, T2 >::Info
Stores some additional information for a bound . . . 942
smtrat::mcsat::InformationGetter . . . 943
smtrat::parser::Instantiator< V, T > . . . 945
smtrat::parser::InstructionHandler . . . 947
Minisat::Int64Range . . . 952
smtrat::IntBlastModule< Settings > . . . 953
smtrat::IntEqModule< Settings >
A module which checks whether the equations contained in the received (linear integer) formula have a solution . . . 980
smtrat::cadcells::representation::util::IntervalCompare< T > . . . 1008
smtrat::covering_ng::IntervalCompare< op >
Sorts interval by their lower bounds . . . 1008
smtrat::mcsat::icp::IntervalPropagation . . . 1009
Minisat::IntOption . . . 1009
Minisat::IntRange . . . 1011
smtrat::is_sample_outside< S > . . . 1011
smtrat::is_sample_outside< IsSampleOutsideAlgorithm::DEFAULT > . . . 1012
smtrat::is_variant< T >
States whether a given type is a `boost::variant` . . . 1012
smtrat::expression::ITEExpression . . . 1012
benchmax::RandomizationAdaptor< T >::iterator . . . 1013
smtrat::ModuleInput::IteratorCompare . . . 1014

benchmax::Jobs
Represents a set of jobs, constructed as the cartesian product of a set of tools and a set of benchmarks . . . 1014
smtrat::parser::KeywordParser
Parses keywords: `:simple_symbol` . . . 1015
Minisat::lbool . . . 1016
smtrat::mcsat::onecell::LDBFilteredAllSelectiveSettings . . . 1017
smtrat::mcsat::onecell::LDBSettings . . . 1017
smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound . . . 1018
smtrat::Module::Lemma . . . 1019
Minisat::LessThan_default< T > . . . 1020
smtrat::cad::projection_compare::level . . . 1020
smtrat::cad::sample_compare::level . . . 1020
smtrat::cad::ProjectionLevelInformation::LevelInfo . . . 1021
smtrat::mcsat::onecellcad::levelwise::LevelwiseCAD . . . 1022
smtrat::LevelWiseInformation< Settings > . . . 1024
smtrat::cad::LiftingTree< Settings > . . . 1027
smtrat::resource::Limiter . . . 1030
smtrat::ICPModule< Settings >::linearVariable . . . 1031
smtrat::mcsat::ClauseChain::Link . . . 1031
Minisat::Lit . . . 1032
benchmax::LocalBackend
This backend simply runs files sequentially on the local machine . . . 1033
smtrat::LOG . . . 1035
smtrat::parser::LogicParser . . . 1036
smtrat::LongFormulaEncoder . . . 1036
smtrat::LRAModule< Settings >
A module which performs the Simplex method on the linear part of it's received formula . . . 1037
smtrat::LVEModule< Settings > . . . 1070
smtrat::Manager
Base class for solvers . . . 1095
Minisat::Map< K, D, H, E > . . . 1104
benchmax::MathSAT
Tool wrapper for MathSAT for SMT-LIB . . . 1106
smtrat::mcsat::fm::MaxSizeComparator
This heuristic chooses the explanation excluding the largest interval . . . 1108
smtrat::MaxSMT< Solver, Strategy > . . . 1109
smtrat::maxsmt::MaxSMTBackend< Solver, Strategy > . . . 1110
smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::FU_MALIK_INCREMENTAL > . . . 1110
smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::LINEAR_SEARCH > . . . 1110
smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::MSU3 > . . . 1111
smtrat::MCBModule< Settings > . . . 1111
smtrat::mcsat::MCSATBackend< Settings > . . . 1137
smtrat::mcsat::MCSATMixin< Settings > . . . 1139
smtrat::expression::simplifier::MergeSimplifier . . . 1145
smtrat::covering_ng::formula::Minimal . . . 1146
benchmax::Minisat
Tool wrapper for a Minisat solver . . . 1147
benchmax::Minisatp
Tool wrapper for the Minisatp solver for pseudo-Boolean problems . . . 1150
smtrat::mcsat::fm::MinSizeComparator
This heuristic chooses the explanation excluding the smallest interval . . . 1152
smtrat::mcsat::fm::MinVarCountComparator
This heuristic tries to minimize the number of variables occuring in the explanation . . . 1153
smtrat::cad::MISGeneration< heuristic > . . . 1153
smtrat::MixedSignEncoder . . . 1154
smtrat::cad::ModelBasedProjection< incrementality, backtracking, Settings > . . . 1156

smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >
This class implements a projection that supports no incrementality and expects backtracking to be in order . . . 1161
smtrat::Module
A base class for all kind of theory solving methods . . . 1167
smtrat::ModuleFactory< Module > . . . 1193
smtrat::ModuleInput
The input formula a module has to consider for it's satisfiability check . . . 1194
smtrat::settings::ModuleSettings . . . 1200
smtrat::Module::ModuleStatistics . . . 1201
smtrat::subtropical::Moment
Represents the normal vector component and the sign variable assigned to a variable in an original constraint . . . 1202
smtrat::MonomialMappingByVariablePool . . . 1202
smtrat::expression::NaryExpression . . . 1203
smtrat::expression::simplifier::NegationSimplifier . . . 1204
smtrat::NewCADModule< Settings > . . . 1205
smtrat::NewCoveringModule< Settings > . . . 1233
benchmax::ssh::Node
Specification of a compuation node for the SSH backend . . . 1263
smtrat::covering_ng::formula::node_ds::Node . . . 1264
smtrat::covering_ng::formula::node_ds::NodeContent . . . 1265
smtrat::mcsat::onecellcad::recursive::NoHeuristic . . . 1266
smtrat::covering_ng::CoveringResult< op >::NONE . . . 1266
smtrat::covering_ng::formula::node_ds::NOT . . . 1266
smtrat::NRAILModule< Settings > . . . 1267
smtrat::parser::NumeralParser
Parses numerals: `(0 | [1-9][0-9]*)` . . . 1293
smtrat::lra::Numeric . . . 1294
smtrat::execution::Objective . . . 1300
Minisat::OccLists< Idx, Vec, Deleted > . . . 1300
smtrat::mcsat::onecellcad::OneCellCAD . . . 1301
benchmax::settings::OperationSettings
Operation settings . . . 1303
smtrat::Optimization< Solver > . . . 1304
Minisat::Option . . . 1305
Minisat::Option::OptionLt . . . 1307
smtrat::covering_ng::formula::node_ds::OR . . . 1307
smtrat::cad::Origin
This class represents one or more origins of some object . . . 1307
smtrat::cad::preprocessor::Origins . . . 1310
Minisat::OutOfMemoryException . . . 1311
smtrat::parser::OutputWrapper . . . 1311
Minisat::Map< K, D, H, E >::Pair . . . 1311
smtrat::mcsat::ParallelExplanation< Backends > . . . 1312
smtrat::parser::ParserSettings . . . 1312
smtrat::parser::ParserState . . . 1312
smtrat::PBGaussModule< Settings > . . . 1316
smtrat::PBPPModule< Settings > . . . 1345
smtrat::PersistentUnionFind . . . 1373
smtrat::PFEModule< Settings > . . . 1375
smtrat::PModule . . . 1402
smtrat::cadcells::operators::properties::poly_additional_root_outside . . . 1428
smtrat::cadcells::operators::properties::poly_irreducible_semi_sgn_inv . . . 1429
smtrat::cadcells::operators::properties::poly_irreducible_sgn_inv . . . 1430
smtrat::cadcells::operators::properties::poly_ord_inv . . . 1430
smtrat::cadcells::operators::properties::poly_ord_inv_base . . . 1431
smtrat::cadcells::operators::properties::poly_pdel . . . 1432

smtrat::cadcells::operators::properties::poly_semi_sgn_inv . . . 1432
smtrat::cadcells::operators::properties::poly_sgn_inv . . . 1433
smtrat::cadcells::representation::util::PolyDelineation . . . 1434
smtrat::cadcells::representation::util::PolyDelineations . . . 1434
smtrat::cad::ProjectionPolynomialInformation::PolyInfo . . . 1434
smtrat::cad::PolynomialComparator< PolynomialGetter > . . . 1435
smtrat::cad::PolynomialLiftingQueue< PolynomialGetter > . . . 1435
smtrat::cadcells::datastructures::PolyPool
A pool for polynomials . . . 1437
smtrat::cadcells::datastructures::detail::PolyProperties . . . 1438
smtrat::cadcells::datastructures::PolyRef
Refers to a polynomial . . . 1439
smtrat::PolyTree . . . 1439
smtrat::PolyTreeContent . . . 1440
smtrat::PolyTreePool . . . 1441
smtrat::cad::Preprocessor . . . 1442
smtrat::cad::PreprocessorSettings . . . 1443
benchmax::settings::PresetSettings . . . 1444
smtrat::PriorityQueue< T, Compare > . . . 1444
smtrat::cad::Projection< incrementality, backtracking, Settings > . . . 1446
smtrat::qe::cad::Projection< Settings > . . . 1451
smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings > . . . 1457
smtrat::cad::Projection< Incrementality::FULL, BT, Settings > . . . 1463
smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >
This class implements a projection that supports no incrementality and expects backtracking to be in order . . . 1468
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 . . . 1474
smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings > . . . 1480
smtrat::cad::projection_compare::ProjectionComparator< Strategy > . . . 1485
smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::D > . . . 1485
smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::ID > . . . 1486
smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::LD > . . . 1486
smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::PD > . . . 1486
smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::SD > . . . 1486
smtrat::cad::projection_compare::ProjectionComparator_impl< Args > . . . 1487
smtrat::cad::ProjectionGlobalInformation . . . 1487
smtrat::cad::ProjectionInformation . . . 1488
smtrat::cad::ProjectionLevelInformation . . . 1490
smtrat::cad::ProjectionOperator . . . 1491
smtrat::cad::ProjectionPolynomialInformation . . . 1492
smtrat::cadcells::datastructures::Projections
Encapsulates all computations on polynomials . . . 1493
smtrat::cadcells::operators::mccallum_filtered_impl::PropertiesSet . . . 1495
smtrat::cadcells::operators::PropertiesSet< Op > . . . 1496
smtrat::cadcells::operators::PropertiesSet< op::mccallum > . . . 1496
smtrat::cadcells::datastructures::PropertiesT< Ts >
Set of properties . . . 1496
smtrat::cadcells::datastructures::PropertiesT< T, Ts... > . . . 1497
smtrat::cadcells::datastructures::PropertiesTContent< T, is_flag > . . . 1497
smtrat::cadcells::datastructures::PropertiesTContent< T, false > . . . 1497
smtrat::cadcells::datastructures::PropertiesTContent< T, true > . . . 1497
smtrat::cadcells::datastructures::property_hash< T > . . . 1497
smtrat::PseudoBoolEncoder
Base class for a PseudoBoolean Encoder . . . 1498
smtrat::PseudoBoolNormalizer . . . 1499
smtrat::parser::QEParser . . . 1500

smtrat::parser::QualifiedIdentifierParser . . . 1501
smtrat::expression::QuantifierExpression . . . 1501
smtrat::parser::QuantifierParser . . . 1502
Minisat::Queue< T > . . . 1502
smtrat::mcsat::icp::QueueEntry . . . 1503
benchmax::RandomizationAdaptor< T >
Provides iteration over a given std::vector in a pseudo-randomized order . . . 1504
smtrat::RationalCapsule . . . 1505
smtrat::parser::RationalPolicies
Specialization of qi::real_policies for a Rational . . . 1506
smtrat::mcsat::onecellcad::RealAlgebraicPoint< Number >
Represent a multidimensional point whose components are algebraic reals . . . 1506
smtrat::mcsat::onecellcad::recursive::RecursiveCAD . . . 1508
smtrat::cad::projection::Reducta< Poly >
Construct the set of reducta of the given polynomial . . . 1512
Minisat::RegionAllocator< T > . . . 1513
smtrat::parser::ErrorHandler::result< typename > . . . 1515
smtrat::cad::preprocessor::ResultantRule . . . 1515
benchmax::Results
Stores results for for whole benchmax run . . . 1516
smtrat::RNSEncoder . . . 1517
smtrat::cadcells::operators::properties::root_ordering_holds . . . 1518
smtrat::cadcells::operators::properties::root_well_def . . . 1519
smtrat::cadcells::datastructures::RootFunction . . . 1519
smtrat::mcsat::arithmetic::RootIndexer< RANT > . . . 1521
smtrat::cad::Sample . . . 1522
smtrat::cad::sample_compare::SampleComparator< Iterator, Strategy > . . . 1524
smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LS > . . . 1525
smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LT > . . . 1525
smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LTA > . . . 1525
smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LTS > . . . 1525
smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LTSA > . . . 1526
smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::S > . . . 1526
smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::T > . . . 1526
smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::TLSA > . . . 1527
smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::TS > . . . 1527
smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::TSA > . . . 1527
smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Type > . . . 1528
smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Value > . . . 1528
smtrat::cad::sample_compare::SampleComparator_impl< It, Args > . . . 1529
smtrat::cadcells::datastructures::SampledDerivation< Properties >
A SampledDerivation is a DelineatedDerivation with a sample and an DelineationInterval w.r.t . 1529
smtrat::SampledDerivationRefCompare . . . 1532
smtrat::cad::SampleIteratorQueue< Iterator, Comparator > . . . 1532
smtrat::covering_ng::sampling< S > . . . 1533
smtrat::sampling< S > . . . 1534
smtrat::covering_ng::sampling< SamplingAlgorithm::LOWER_UPPER_BETWEEN_SAMPLING > . . . 1534
smtrat::sampling< SamplingAlgorithm::LOWER_UPPER_BETWEEN_SAMPLING > . . . 1534
smtrat::SATModule< Settings >
Implements a module performing DPLL style SAT checking . . . 1535
smtrat::parser::ScriptParser< Callee > . . . 1567
smtrat::parser::ParserState::ScriptScope . . . 1569
smtrat::mcsat::onecellcad::Section
Represent a cell's (closed-interval-boundary) component along th k-th axis . . . 1570
smtrat::onecellcad::recursive::Section
Represent a cell's closed-interval-boundary object along one single axis by an irreducible, multivariate polynomial of level k . . . 1570
smtrat::mcsat::onecellcad::levelwise::SectionHeuristic1 . . . 1571

smtrat::mcsat::onecellcad::levelwise::SectionHeuristic2 . . . 1571
smtrat::mcsat::onecellcad::levelwise::SectionHeuristic3 . . . 1571
smtrat::mcsat::onecellcad::Sector
Represent a cell's open-interval boundary object along one single axis by two irreducible, multivariate polynomials of level k . . . 1572
smtrat::onecellcad::recursive::Sector
Represent a cell's open-interval boundary object along one single axis by two irreducible, multivariate polynomials of level k . . . 1572
smtrat::mcsat::onecellcad::levelwise::SectorHeuristic1 . . . 1573
smtrat::mcsat::onecellcad::levelwise::SectorHeuristic2 . . . 1573
smtrat::mcsat::onecellcad::levelwise::SectorHeuristic3 . . . 1574
smtrat::subtropical::Separator
Represents the class of all original constraints with the same left hand side after a normalization 1574
smtrat::mcsat::SequentialAssignment< Backends > . . . 1575
smtrat::mcsat::SequentialExplanation< Backends > . . . 1575
benchmax::settings::Settings
Generic class to manage runtime settings . . . 1575
smtrat::settings::Settings . . . 1575
smtrat::SettingsComponents . . . 1575
benchmax::SettingsParser
Generic class to manage settings parsing using boost::program_options . . . 1575
smtrat::SettingsParser . . . 1575
smtrat::CNFerModule::SettingsType . . . 1576
smtrat::parser::SExpressionParser . . . 1576
smtrat::parser::SExpressionSequence< T > . . . 1576
smtrat::ShortFormulaEncoder . . . 1577
benchmax::simple_parser . . . 1578
smtrat::parser::Theories::SimpleSortAdder
Helper functor for addSimpleSorts() method . . . 1579
smtrat::parser::SimpleSymbolParser
Parses symbols: `simple_symbol` | `quoted_symbol` where . . . 1580
smtrat::expression::simplifier::Simplifier . . . 1581
smtrat::expression::simplifier::SimplifierChainCaller< chainID > . . . 1581
smtrat::expression::simplifier::SimplifierChainCaller< 0 > . . . 1581
smtrat::expression::simplifier::SingletonSimplifier . . . 1582
smtrat::cad::sample_compare::size . . . 1583
smtrat::parser::Skipper . . . 1583
benchmax::SlurmBackend
Backend for the Slurm workload manager . . . 1584
benchmax::settings::SlurmBackendSettings
Settings for the Slurm backend . . . 1586
smtrat::parser::SMTLIBParser . . . 1587
benchmax::SMTRAT
Tool wrapper for SMT-RAT for SMT-LIB . . . 1590
benchmax::SMTRAT_Analyzer
Tool wrapper for SMT-RAT for SMT-LIB . . . 1593
benchmax::SMTRAT_OPB
Adapts the SMTRAT solver for OPB files . . . 1595
smtrat::execution::SoftAssertion . . . 1598
smtrat::settings::SolverSettings . . . 1598
smtrat::parser::SortedVariableParser . . . 1599
smtrat::parser::SortParser . . . 1600
smtrat::parser::SpecConstantParser . . . 1600
smtrat::SplitSOSModule< Settings > . . . 1601
benchmax::SSHBackend . . . 1628
benchmax::settings::SSHBackendSettings
Settings for SSH backend . . . 1630

benchmax::ssh::SSHConnection
A wrapper class that manages a single SSH connection as specified in a Node object (with all its channels) . . . 1631
smtrat::vs::State . . . 1633
smtrat::StaticUnionFind . . . 1657
smtrat::statistics::StatisticsSettings . . . 1658
smtrat::StrategyGraph . . . 1658
Minisat::StringOption . . . 1660
smtrat::parser::StringParser
Parses strings: "."+" with escape sequences \\" and \\\\ . . . 1661
smtrat::STropModule< Settings > . . . 1661
benchmax::slurm::SubmitfileProperties
All properties needed to create a submit file . . . 1688
smtrat::vs::Substitution . . . 1689
smtrat::vs::substitutionPointerEqual . . . 1693
smtrat::vs::substitutionPointerHash . . . 1693
smtrat::cadcells::datastructures::SymbolicInterval
A symbolic interal whose bounds are defined by indexed root expressions . . . 1693
smtrat::parser::SymbolParser . . . 1695
smtrat::SymmetryModule< Settings > . . . 1696
smtrat::lra::Tableau< Settings, T1, T2 > . . . 1722
smtrat::lra::TableauEntry< T1, T2 > . . . 1740
smtrat::cadcells::datastructures::TaggedIndexedRoot . . . 1743
smtrat::mcsat::onecellcad::TagPoly
Tagged Polynomials . . . 1744
smtrat::Task . . . 1744
smtrat::parser::TermParser . . . 1745
smtrat::mcsat::vs::helper::TestCandidate . . . 1747
smtrat::parser::Theories
Combines all implemented theories and provides a single interface to interact with all theories at once . . . 1747
smtrat::parser::TheoryError . . . 1750
smtrat::mcsat::TheoryLevel . . . 1751
smtrat::Module::TheoryPropagation . . . 1752
smtrat::TheoryVarSchedulerStatic< vot >
Schedules theory variables statically . . . 1753
smtrat::ThreadPool . . . 1757
smtrat::cad::debug::TikzBasePrinter . . . 1757
smtrat::cad::debug::TikzDAGPrinter . . . 1758
smtrat::cad::debug::TikzHistoryPrinter . . . 1760
smtrat::cad::debug::TikzTreePrinter . . . 1761
benchmax::Tool
Base class for any tool . . . 1762
benchmax::settings::ToolSettings
Tool-related settings . . . 1766
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 . . . 1767
smtrat::TotalizerTree . . . 1769
smtrat::covering_ng::formula::node_ds::TRUE . . . 1769
smtrat::cad::projection_compare::type . . . 1769
smtrat::cad::sample_compare::type . . . 1769
smtrat::UFCegarModule< Settings > . . . 1770
smtrat::expression::UnaryExpression . . . 1797
smtrat::cad::debug::UnifiedData . . . 1797
smtrat::parser::types::UninterpretedTheory
Types of the theory of equalities and uninterpreted functions . . . 1798

smtrat::parser::UninterpretedTheory
Implements the theory of equalities and uninterpreted functions . . . 1799
smtrat::UnionFindInterface< T, Implementation > . . . 1802
smtrat::UnionFindModule< Settings > . . . 1803
smtrat::SATModule< Settings >::UnorderedClauseLookup::UnorderedClauseHasher . . . 1830
smtrat::UnsatCore< Solver, Strategy > . . . 1830
smtrat::unsatcore::UnsatCoreBackend< Solver, Strategy > . . . 1831
smtrat::unsatcore::UnsatCoreBackend< Solver, UnsatCoreStrategy::ModelExclusion >
Implements an easy strategy to obtain an unsatisfiable core . . . 1831
smtrat::vs::unsignedTripleCmp . . . 1832
smtrat::parser::UserFunctionInstantiator . . . 1833
smtrat::validation::ValidationCollector . . . 1834
smtrat::validation::ValidationPoint . . . 1834
smtrat::validation::ValidationPrinter< SOF > . . . 1835
smtrat::validation::ValidationSettings . . . 1835
smtrat::lra::Value< T > . . . 1836
smtrat::lra::Variable< T1, T2 > . . . 1843
smtrat::vb::Variable< T >
Class for a variable . . . 1854
smtrat::vb::VariableBounds< T >
Class to manage the bounds of a variable . . . 1857
smtrat::VariableCapsule . . . 1864
smtrat::mcsat::variableordering::VariableIDs . . . 1865
smtrat::VariableRewriteRule . . . 1865
smtrat::parser::conversion::VariantConverter< Res >
Converts variants to some type using the Converter class . . . 1866
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` . . . 1868
smtrat::parser::conversion::VariantVariantConverter< Res >
Converts variants to another variant type not using the Converter class . . . 1870
smtrat::VarSchedulerBase
Base class for variable schedulers . . . 1871
smtrat::VarSchedulerFixedRandom . . . 1874
smtrat::VarSchedulerMcsatActivityPreferTheory< vot >
Activity-based scheduler preferring initially theory variables . . . 1877
smtrat::VarSchedulerMcsatBase
Base class for all MCSAT variable scheduler . . . 1880
smtrat::VarSchedulerMcsatBooleanFirst< vot >
Variable scheduling that all decides Boolean variables first before deciding any theory variable 1884
smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >
Variable scheduling that all decides theory variables first before deciding any Boolean variable 1887
smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >
Decides only Constraints occuring in clauses that are univariate in the current theory variable while the theory ordering is static . . . 1891
smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >
Decides only Constraints univariate in the current theory variable while the theory ordering is static . . . 1895
smtrat::VarSchedulerMinisat
Minisat's activity-based variable scheduling . . . 1898
smtrat::VarSchedulerRandom
Random scheduler . . . 1902
smtrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic >
Scheduler for SMT, implementing theory guided heuristics . . . 1905
Minisat::vec< T > . . . 1908
smtrat::parser::conversion::VectorVariantConverter< Res >
Converts a vector of variants to a vector of some type using the Converter class . . . 1911

smtrat::subtropical::Vertex
Represents a term of an original constraint and assigns him a rating variable if a weak separator is searched . . . 1912
smtrat::VSModule< Settings > . . . 1913
Minisat::Watcher
[Minisat related code] . . . 1942
smtrat::ICPModule< Settings >::weights . . . 1943
benchmax::XMLWriter
Writes results to a xml file . . . 1944
smtrat::covering_ng::formula::node_ds::XOR . . . 1944
benchmax::Z3
Tool wrapper for Z3 for SMT-LIB . . . 1944

0.14 Namespace Documentation

0.14.1 benchmax Namespace Reference

Namespaces

- settings
- slurm
- ssh

Data Structures

- class Backend

  Base class for all backends.
- class CondorBackend

  Backend for the HTCondor batch system.
- class RandomizationAdaptor

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

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

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

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

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

  Results for a single benchmark run.
- class Database

  Dummy database that effectively disables storing to database. Set BENCHMAX_DATABASE to actually use a database.
- class DBAL
- class Results

  Stores results for for whole benchmax run.
- class XMLWriter

  Writes results to a xml file.
- class SettingsParser

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

  Tool wrapper for MathSAT for SMT-LIB.

- class Minisat

  Tool wrapper for a Minisat solver.
- class Minisatp

  Tool wrapper for the Minisatp solver for pseudo-Boolean problems.
- class simple_parser
- class SMTRAT

  Tool wrapper for SMT-RAT for SMT-LIB.
- class SMTRAT_Analyzer

  Tool wrapper for SMT-RAT for SMT-LIB.
- class SMTRAT_OPB

  Adapts the SMTRAT solver for OPB files.
- class Tool

  Base class for any tool.
- class Z3

  Tool wrapper for Z3 for SMT-LIB.

Typedefs

- using Job = std::pair< const Tool *, std::filesystem::path >

  Typedef for a job, consisting of a tool and an input file.
- using ToolPtr = std::unique_ptr< Tool >

  A std::unique_ptr to a Tool.
- using Tools = std::vector< ToolPtr >

  A vector of ToolPtr.

Functions

- template<typename Backend >
  void execute_backend (const std::string &name, const Jobs &jobs)
- void run_backend (const std::string &backend, const Jobs &jobs)

  Runs a given backend on a list of tools and benchmarks.
- const auto & settings_slurm ()

  Return the Slurm settings.
- const auto & settings_ssh ()

  Return the SSH settings.
- BenchmarkSet loadBenchmarks ()

  Loads benchmark files from directory specified in settings.
- const auto & settings_benchmarks ()

  Return the benchmark settings.
- void logging_configure ()

  Configure default logging for benchmax.
- void logging_quiet ()

  Configure quiet logging for benchmax.
- void logging_verbose ()

  Configure verbose logging for benchmax.
- std::ostream & operator<< (std::ostream &os, const BenchmarkResult &results)

  Streaming operator for BenchmarkResult.
- const auto & settings_preset ()

  Return the Slurm settings.
- template<typename T >
  const auto & settings_get (const std::string &name)

  Helper function to obtain settings by name and type.

- const auto & settings_core ()

  Retrieved core settings.
- const auto & settings_operation ()

  Retrieved operation settings.
- bool is_system_lib (std::string path)
- std::ostream & operator<< (std::ostream &os, const Tool &tool)

  Streaming operator for a generic tool.
- template<typename T >
  void createTools (const std::vector< std::filesystem::path > &arguments, Tools &tools)

  Create tools of a given type T from a list of binaries and store them in tools.
- Tools loadTools ()

  Load all tools from the tool settings.
- const auto & settings_tools ()

  Returns the tool settings.
- int call_program (const std::string &commandline, std::string &stdout, bool print_to_stdout=false)

  Runs an external program from some command line and records the output to stdout.
- std::filesystem::path common_prefix (const std::filesystem::path &p1, const std::filesystem::path &p2)

  Computes the common prefix of two paths.
- std::filesystem::path common_prefix (const std::vector< std::filesystem::path > &s, bool skip_last=true)

  Computes the common prefix of a list of paths.
- std::filesystem::path common_prefix (const std::initializer_list< std::vector< std::filesystem::path >> &s, bool skip_last=true)

  Computes the common prefix of multiple lists of paths.
- std::filesystem::path remove_prefix (const std::filesystem::path &s, const std::filesystem::path &prefix)

  Remove a prefix from a path.
- bool is_extension (const std::filesystem::path &path, const std::string &extension)

  Checks whether the extension of the filename is as specified.
- std::size_t parse_peak_memory (const std::string &output)

0.14.1.1 Typedef Documentation

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

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

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

A std::unique_ptr to a Tool.

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

A vector of ToolPtr.

0.14.1.2 Function Documentation

0.14.1.2.1 call_program()

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

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

Parameters

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

Returns

Exit code of the program.

0.14.1.2.2 common_prefix() [1/3]

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

Computes the common prefix of two paths.

0.14.1.2.3 common_prefix() [2/3]

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

Computes the common prefix of multiple lists of paths.
skip_last specifies whether the last part if the path (usually the filename) may be part of the prefix.

0.14.1.2.4 common_prefix() [3/3]

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

Computes the common prefix of a list of paths.
skip_last specifies whether the last part if the path (usually the filename) may be part of the prefix.

0.14.1.2.5 createTools()

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

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

0.14.1.2.6 execute_backend()

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

0.14.1.2.7 is_extension()

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

Checks whether the extension of the filename is as specified.

0.14.1.2.8 is_system_lib()

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

0.14.1.2.9 loadBenchmarks()

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

Loads benchmark files from directory specified in settings.

0.14.1.2.10 loadTools() `Tools benchmax::loadTools ( )`

Load all tools from the tool settings.

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

Configure default logging for benchmax.

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

Configure quiet logging for benchmax.

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

Configure verbose logging for benchmax.

0.14.1.2.14 operator<<() [1/2]

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

Streaming operator for BenchmarkResult.

0.14.1.2.15 operator<<() [2/2]

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

Streaming operator for a generic tool.

0.14.1.2.16 parse_peak_memory()

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

0.14.1.2.17 remove_prefix()

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

Remove a prefix from a path.

0.14.1.2.18 run_backend()

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

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

Parameters

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

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

Return the benchmark settings.

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

Retrieved core settings.

0.14.1.2.21 settings_get() `template<typename T >`

```
const auto& benchmax::settings_get (
            const std::string & name ) [inline]
```

Helper function to obtain settings by name and type.

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

Retrieved operation settings.

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

Return the Slurm settings.

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

Return the Slurm settings.

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

Return the SSH settings.

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

Returns the tool settings.

0.14.2 benchmax::settings Namespace Reference

Data Structures

- struct SlurmBackendSettings

  Settings for the Slurm backend.
- struct SSHBackendSettings

  Settings for SSH backend.
- struct BenchmarkSettings

  Settings for benchmarks.
- struct PresetSettings
- struct CoreSettings

  Core settings.
- struct OperationSettings

  Operation settings.
- struct Settings

  Generic class to manage runtime settings.
- struct ToolSettings

  Tool-related settings.

Functions

- void registerSlurmBackendSettings (SettingsParser *parser)

  Registers Slurm settings with the settings parser.
- void registerSSHBackendSettings (SettingsParser *parser)

  Registers SSH settings with settings parser.

- bool finalize_benchmark_settings (BenchmarkSettings &s, const boost::program_options::variables_map &)

  Postprocess benchmark settings.
- void registerBenchmarkSettings (SettingsParser *parser)

  Registers benchmark settings with the settings parser.
- bool finalize_preset_settings (PresetSettings &s, boost::program_options::variables_map &values)

  Postprocess preset settings.
- void registerPresetSettings (SettingsParser *parser)

  Registers preset settings with the settings parser.
- bool finalize_tool_settings (ToolSettings &s, const boost::program_options::variables_map &)

  Postprocess settings to compute common prefix.
- void registerToolSettings (SettingsParser *parser)

  Registers tool settings with the settings parser.

0.14.2.1 Function Documentation

0.14.2.1.1 finalize_benchmark_settings()

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

Postprocess benchmark settings.

0.14.2.1.2 finalize_preset_settings()

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

Postprocess preset settings.

0.14.2.1.3 finalize_tool_settings()

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

Postprocess settings to compute common prefix.

0.14.2.1.4 registerBenchmarkSettings()

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

Registers benchmark settings with the settings parser.

0.14.2.1.5 registerPresetSettings()

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

Registers preset settings with the settings parser.

0.14.2.1.6 registerSlurmBackendSettings()

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

Registers Slurm settings with the settings parser.

0.14.2.1.7 registerSSHBackendSettings()

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

Registers SSH settings with settings parser.

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

Registers tool settings with the settings parser.

0.14.3 benchmax::slurm Namespace Reference

Data Structures

- struct ArchiveProperties

  All properties needed to archive log files.
- struct SubmitfileProperties

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

  All properties needed to create a submit file.

Functions

- void archive_log_files (const ArchiveProperties &p)

  Put all log files into an archive.
- std::vector< fs::path > collect_result_files (const fs::path &basedir)

  Collects all result files in the given base directory for this job id.
- std::string generate_submit_file (const SubmitfileProperties &p)

  Generate a submit file for Slurm with the given properties.
- std::string generate_submit_file_chunked (const ChunkedSubmitfileProperties &p)
- int parse_job_id (const std::string &output)

  Parses the job id from the output of `sbatch`.
- std::string parse_result_info (const std::string &content, const std::string &name)

  Parse a single result information from the output.
- void remove_log_files (const std::vector< fs::path > &files, bool remove)

  Remove the given list of files.
- void clear_directory (const fs::path &basedir)

  Clear log files from directory.
- bool is_job_finished (int jobid)

  Checks if the given job is finished.
- template<typename Jobs >
  void generate_jobs_file (const std::string &filename, std::pair< std::size_t, std::size_t > range, const Jobs &jobs)

0.14.3.1 Function Documentation

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

Put all log files into an archive.

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

Clear log files from directory.

Parameters

basedir	Base directory to search in.

0.14.3.1.3 collect_result_files()

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

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

Parameters

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

Returns

List of result files.

0.14.3.1.4 generate_jobs_file()

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

0.14.3.1.5 generate_submit_file()

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

Generate a submit file for Slurm with the given properties.

Parameters

p	Collection of all information needed.

Returns

The filename of the submit file.

0.14.3.1.6 generate_submit_file_chunked()

```
std::string benchmax::slurm::generate_submit_file_↩
chunked (
    const ChunkedSubmitfileProperties & p )
```

0.14.3.1.7 is_job_finished()

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

Checks if the given job is finished.

Parameters

jobid	The job id.

Returns

True if the job is completed.

0.14.3.1.8 parse_job_id()

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

Parses the job id from the output of `sbatch`.

Parameters

output	Output of `sbatch`.

Returns

Slurm job id.

0.14.3.1.9 parse_result_info()

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

Parse a single result information from the output.

Parameters

content	Result information output.
name	Information to be parsed.

Returns

Information specified in the output.

0.14.3.1.10 remove_log_files()

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

Remove the given list of files.

Parameters

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

0.14.4 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).

0.14.5 carl Namespace Reference

0.14.6 Minisat Namespace Reference

Data Structures

- class RegionAllocator

- class Heap
- struct Hash
- struct Equal
- struct DeepHash
- struct DeepEqual
- class Map
- class Option
- struct IntRange
- struct Int64Range
- struct DoubleRange
- class DoubleOption
- class IntOption
- class StringOption
- class BoolOption
- class Queue
- struct Lit
- class lbool
- class Clause
- class ClauseAllocator
- class OccLists
- class CMap
- struct Watcher

  [Minisat related code]
- struct LessThan_default
- class vec
- class OutOfMemoryException

Typedefs

- typedef int Var
- typedef RegionAllocator< uint32_t >::Ref CRef

Functions

- template<class V , class T >
  static void remove (V &ts, const T &t)
- template<class V , class T >
  static bool find (V &ts, const T &t)
- template<class T >
  static void copy (const T &from, T &to)
- template<class T >
  static void copy (const vec< T > &from, vec< T > &to, bool append=false)
- template<class T >
  static void append (const vec< T > &from, vec< T > &to)
- static uint32_t hash (uint32_t x)
- static uint32_t hash (uint64_t x)
- static uint32_t hash (int32_t x)
- static uint32_t hash (int64_t x)
- void printUsageAndExit (int argc, char **argv, bool verbose=false)
- void setUsageHelp (const char *str)
- void setHelpPrefixStr (const char *str)
- Lit mkLit (Var var, bool sign=false)
- Lit operator~ (Lit p)
- Lit operator^ (Lit p, bool b)
- bool sign (Lit p)

- int var (Lit p)
- Lit neg (Lit p)
- int toInt (Var v)
- int toInt (Lit p)
- Lit toLit (int i)
- std::ostream & operator<< (std::ostream &os, const Lit &l)
- int toInt (lbool l)
- lbool toLbool (int v)
- std::ostream & operator<< (std::ostream &os, Minisat::lbool b)
- std::ostream & operator<< (std::ostream &os, const Minisat::Clause &c)
- template<class T , class LessThan >
  void selectionSort (T *array, int size, LessThan lt)
- template<class T >
  static void selectionSort (T *array, int size)
- template<class T , class LessThan >
  void sort (T *array, int size, LessThan lt)
- template<class T >
  static void sort (T *array, int size)
- template<class T , class LessThan >
  void sort (vec< T > &v, LessThan lt)
- template<class T >
  void sort (vec< T > &v)
- template<typename T >
  std::ostream & operator<< (std::ostream &os, const vec< T > &v)
- template<> std::ostream & operator<< (std::ostream &os, const vec< char > &v)
- static void * xrealloc (void *ptr, size_t size)

Variables

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

0.14.6.1 Typedef Documentation

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

0.14.6.1.2 Var `typedef int Minisat::Var`

0.14.6.2 Function Documentation

0.14.6.2.1 append()

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

0.14.6.2.2 copy() [1/2]

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

0.14.6.2.3 copy() [2/2]

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

0.14.6.2.4 find()

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

0.14.6.2.5 hash() [1/4]

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

0.14.6.2.6 hash() [2/4]

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

0.14.6.2.7 hash() [3/4]

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

0.14.6.2.8 hash() [4/4]

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

0.14.6.2.9 mkLit()

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

0.14.6.2.10 neg()

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

0.14.6.2.11 operator<<() [1/5]

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

0.14.6.2.12 operator<<() [2/5]

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

0.14.6.2.13 operator<<() [3/5]

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

0.14.6.2.14 operator<<() [4/5]

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

0.14.6.2.15 operator<<() [5/5]

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

0.14.6.2.16 operator^()

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

0.14.6.2.17 operator~()

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

0.14.6.2.18 printUsageAndExit()

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

0.14.6.2.19 remove()

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

0.14.6.2.20 selectionSort() [1/2]

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

0.14.6.2.21 selectionSort() [2/2]

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

0.14.6.2.22 setHelpPrefixStr()

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

0.14.6.2.23 setUsageHelp()

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

0.14.6.2.24 sign()

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

0.14.6.2.25 sort() [1/4]

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

0.14.6.2.26 sort() [2/4]

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

0.14.6.2.27 sort() [3/4]

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

0.14.6.2.28 sort() [4/4]

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

0.14.6.2.29 toInt() [1/3]

```
int Minisat::toInt (
    lbool l )  [inline]
```

0.14.6.2.30 toInt() [2/3]

```
int Minisat::toInt (
    Lit p )  [inline]
```

0.14.6.2.31 toInt() [3/3]

```
int Minisat::toInt (
    Var v )  [inline]
```

0.14.6.2.32 toLbool()

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

0.14.6.2.33 toLit()

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

0.14.6.2.34 var()

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

0.14.6.2.35 xrealloc()

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

0.14.6.3 Variable Documentation

0.14.6.3.1 CONFLICT_CLAUSE

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

0.14.6.3.2 CRef_Lazy

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

0.14.6.3.3 CRef_TPropagation

```
const CRef Minisat::CRef_TPropagation = RegionAllocator<uint32_↩
t>::Ref_Undef - 2
```

0.14.6.3.4 CRef_Undef

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

0.14.6.3.5 LEMMA_CLAUSE

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

0.14.6.3.6 lit_Error

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

0.14.6.3.7 lit_Undef

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

0.14.6.3.8 NORMAL_CLAUSE

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

0.14.6.3.9 nprimes

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

0.14.6.3.10 PERMANENT_CLAUSE

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

0.14.6.3.11 primes

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

0.14.7 smtrat Namespace Reference

Class to create the formulas for axioms.

Namespaces

- analyzer
- cad
- cadcells

  A framework for sample-based CAD algorithms.
- compile_information

  Compile time generated information about compiler and system version.
- covering_ng
- execution
- expression
- groebner
- helper
- icp
- lra
- lve
- maxsmt

  Contains strategy implementations for max SMT computations.
- mcsat
- onecellcad
- options_detail
- parseformula
- parser
- qe
- resource
- sat
- settings
- statistics
- subtropical

  Implements data structures and encodings for the subtropical method.
- unsatcore

  Contains strategy implementations for unsat core computations.
- validation
- vb
- vs

Data Structures

- struct is_variant

  States whether a given type is a `boost::variant`.
- struct **is_variant**< **boost::variant**< **BOOST_VARIANT_ENUM_PARAMS(U)**> >

  States that `boost::variant` *is indeed a* `boost::variant`.
- struct **is_variant**< **const boost::variant**< **BOOST_VARIANT_ENUM_PARAMS(U)**> >

  States that `const boost::variant` *is indeed a* `boost::variant`.
- class VariantMap

  This class is a specialization of `std::map` *where the keys are of arbitrary type and the values are of type* `boost↩::variant`.
- struct CMakeOptionPrinter
- class Executor
- class PriorityQueue
- class DynamicPriorityQueue
- class SettingsComponents
- class SettingsParser
- class MaxSMT

- class BEModule
- class BVDirectEncoder
- class BVModule
- class CNFerModule
- class CoCoAGBModule
- class CoveringNGModule
- class Bimap

  Container that stores expensive to construct objects and allows the fast lookup with respect to two independent keys within the objects.
- class CSplitModule
- class CubeLIAModule
- class CurryModule
- class EMModule
- class ESModule
- class FouMoModule

  A module which applies the Fourier-Motzkin algorithm.
- class FPPModule
- class GBModule

  A solver module based on Groebner basis.
- class GBModuleState

  A class to save the current state of a GBModule.
- class InequalitiesTable

  Datastructure for the GBModule.
- class VariableRewriteRule
- class GBPPModule
- class ForwardHyperGraph

  This class implements a forward hypergraph.
- struct CycleEnumerator

  This class encapsulates an algorithm for enumerating all cycles.
- class ICEModule
- class ICPModule
- class IncWidthModule
- class BVAnnotation
- class AnnotatedBVTerm
- class BlastedPoly
- class BlastedConstr
- class ConstrTree
- class ElementWithOrigins
- class CollectionWithOrigins
- class IntBlastModule
- class PolyTree
- class PolyTreeContent
- class PolyTreePool
- class IntEqModule

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

  A module which performs the Simplex method on the linear part of it's received formula.
- class LVEModule
- class MCBModule
- class NewCADModule
- class Backend
- struct SampledDerivationRefCompare
- class LevelWiseInformation

- class NewCoveringModule
- struct sampling
- struct is_sample_outside
- struct sampling< SamplingAlgorithm::LOWER_UPPER_BETWEEN_SAMPLING >
- struct is_sample_outside< IsSampleOutsideAlgorithm::DEFAULT >
- class RationalCapsule
- class VariableCapsule
- class AxiomFactory
- class LOG
- class MonomialMappingByVariablePool
- class NRAILModule
- class PBGaussModule
- class CardinalityEncoder
- class ExactlyOneCommanderEncoder
- class LongFormulaEncoder
- class MixedSignEncoder
- class PseudoBoolEncoder

  Base class for a PseudoBoolean Encoder.
- class PseudoBoolNormalizer
- class RNSEncoder
- class ShortFormulaEncoder
- class TotalizerEncoder

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

  Base class for all MCSAT variable scheduler.
- class TheoryVarSchedulerStatic

  Schedules theory variables statically.
- class VarSchedulerMcsatBooleanFirst

  Variable scheduling that all decides Boolean variables first before deciding any theory variable.
- class VarSchedulerMcsatTheoryFirst

  Variable scheduling that all decides theory variables first before deciding any Boolean variable.
- class VarSchedulerMcsatUnivariateConstraintsOnly

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

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

  Activity-based scheduler preferring initially theory variables.
- class SATModule

  Implements a module performing DPLL style SAT checking.
- class VarSchedulerBase

  Base class for variable schedulers.
- class VarSchedulerMinisat

  Minisat's activity-based variable scheduling.
- class VarSchedulerFixedRandom
- class VarSchedulerRandom

  Random scheduler.
- class VarSchedulerSMTTheoryGuided

  Scheduler for SMT, implementing theory guided heuristics.

- class SplitSOSModule
- class STropModule
- class SymmetryModule
- class UFCegarModule
- struct EQGraph
- struct UnionFindInterface
- struct StaticUnionFind
- struct PersistentUnionFind
- struct Backtrackable
- class UnionFindModule
- class VSModule
- class Optimization
- class Manager

  Base class for solvers.
- struct Branching

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

  A base class for all kind of theory solving methods.
- class FormulaWithOrigins

  Stores a formula along with its origins.
- class ModuleInput

  The input formula a module has to consider for it's satisfiability check.
- class PModule
- struct AbstractModuleFactory
- struct ModuleFactory
- class BackendLink
- class StrategyGraph
- class Task
- class BackendSynchronisation
- class ThreadPool
- class UnsatCore

Typedefs

- using ModelVariable = carl::ModelVariable
- using ModelValue = carl::ModelValue< Rational, Poly >
- using Model = carl::Model< Rational, Poly >
- using ModelSubstitution = carl::ModelSubstitution< Rational, Poly >
- using ModelPolynomialSubstitution = carl::ModelPolynomialSubstitution< Rational, Poly >
- using InfinityValue = carl::InfinityValue
- using SqrtEx = carl::SqrtEx< Poly >
- using MultivariateRootT = carl::MultivariateRoot< Poly >
- using DoubleInterval = carl::Interval< double >
- using RationalInterval = carl::Interval< Rational >
- using EvalDoubleIntervalMap = std::map< carl::Variable, DoubleInterval >
- using EvalRationalIntervalMap = std::map< carl::Variable, RationalInterval >
- using ObjectiveValues = std::vector< std::pair< std::variant< Poly, std::string >, Model::mapped_type > >
- using Statistics = carl::statistics::Statistics
- using Conditionals = std::vector< std::atomic_bool * >

  A vector of atomic bool pointers.
- using Rational = mpq_class
- using Integer = carl::IntegralType< Rational >::type
- using TermT = carl::Term< Rational >
- using Poly = carl::MultivariatePolynomial< Rational >

- using Factorization = std::map< Poly, carl::uint >
- using ConstraintT = carl::Constraint< Poly >
- using ConstraintsT = carl::Constraints< Poly >
- using VariableAssignmentT = carl::VariableAssignment< Poly >
- using VariableComparisonT = carl::VariableComparison< Poly >
- using FormulaT = carl::Formula< Poly >
- using FormulasT = carl::Formulas< Poly >
- using FormulaSetT = carl::FormulaSet< Poly >
- using FormulasMultiT = std::multiset< FormulaT >
- using RationalAssignment = carl::Assignment< Rational >
- using RAN = carl::IntRepRealAlgebraicNumber< Rational >
- using thread_priority = std::pair< std::size_t, std::size_t >
- typedef expression::Expression Expression
- typedef expression::Expressions Expressions
- typedef std::pair< FormulaT, std::shared_ptr< std::vector< FormulaT > > > SingleFormulaOrigins
- typedef std::map< FormulaT, std::shared_ptr< std::vector< FormulaT > > > FormulaOrigins
- typedef std::map< carl::Variable, std::pair< std::vector< SingleFormulaOrigins >, std::vector< SingleFormulaOrigins > > > VariableUpperLower
- typedef std::set< icp::ContractionCandidate *, icp::contractionCandidateComp > ContractionCandidates
- typedef std::map< FormulaT, std::shared_ptr< std::vector< FormulaT > > > Formula_Origins
- typedef std::unordered_map< carl::Variable, carl::Monomial::Arg >::iterator MonomialMapIterator
- typedef std::unordered_map< carl::Variable, carl::Monomial::Arg > MonomialMap
- using Type = carl::UVariable
- template<typename Impl >
  using EQClasses = UnionFindInterface< Type, Impl >
- using StaticClasses = EQClasses< StaticUnionFind >
- using BacktrackableClasses = EQClasses< Backtrackable< PersistentUnionFind > >
- using Classes = BacktrackableClasses
- typedef bool(* ConditionEvaluation) (carl::Condition)
- typedef std::function< bool(carl::Condition)> ConditionFunction
- using Priority = std::vector< std::size_t >

Enumerations

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

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

Functions

- int handle_basic_options (const SettingsParser &parser)
- const auto & settings_parser ()
- bool parseSMT2File (parser::InstructionHandler &handler, bool queueInstructions, std::istream &input)
- int convert_to_cnf_dimacs (const std::string &, const std::string &)
- int convert_to_cnf_smtlib (const std::string &, const std::string &)
- std::ostream & operator<< (std::ostream &os, CMakeOptionPrinter cmop)

- constexpr CMakeOptionPrinter CMakeOptions (bool advanced=false) noexcept
- template<typename Executor >
  bool parseInput (const std::string &pathToInputFile, Executor &e, bool &queueInstructions)
- template<typename Executor >
  int executeFile (const std::string &pathToInputFile, Executor &e)

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

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

  Loads the smt2 file specified in filename and runs the preprocessing strategy.
- const auto & settings_analyzer ()
- analyzer::AnalyzerStatistics & analyze_formula (const FormulaT &f)
- template<typename TT , typename C >
  std::ostream & operator<< (std::ostream &os, const PriorityQueue< TT, C > &pq)
- const settings::Settings & Settings ()
- const auto & settings_core ()
- const auto & settings_solver ()
- const auto & settings_module ()
- template<typename T >
  auto & statistics_get (const std::string &name)
- const auto & settings_statistics ()
- bool is_sat (Answer a)
- std::ostream & operator<< (std::ostream &os, Answer a)
- auto & validation_get (const std::string &channel, const std::string &file, int line)
- const auto & settings_validation ()
- void visiting (const Expression &expr)
- void testExpression ()
- std::ostream & operator<< (std::ostream &os, MaxSMTStrategy ucs)
- 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)
- Rational factorial (Rational n)
- Rational factorial (std::size_t)
- std::ostream & operator<< (std::ostream &os, Module::LemmaType lt)
- template<typename AnnotationType >
  void annotateFormula (const FormulaT &, const std::vector< AnnotationType > &)
- std::ostream & operator<< (std::ostream &os, const ModuleInput &mi)
- bool isCondition (carl::Condition _condition)
- template<typename Module >
  std::ostream & operator<< (std::ostream &os, const ModuleFactory< Module > &mf)
- std::ostream & operator<< (std::ostream &os, UnsatCoreStrategy ucs)

Variables

- static const Model EMPTY_MODEL = Model()
- static constexpr int COMMANDER_GROUP_SIZE = 4

0.14.7.1 Detailed Description

Class to create the formulas for axioms.
Class to create a substitution object.
Class to create a state object.
Class to create a condition object.

Author

Aklima Zaman

Since

2018-09-24

Version

2018-09-24

Author

Florian Corzilius

Since

2010-07-26

Version

2011-06-20

Author

Florian Corzilius

Since

2010-07-26

Version

2011-12-05

Author

Florian Corzilius

Since

2010-05-11

Version

2013-10-24

Author

Florian Corzilius

Since

2010-05-11

Version

2013-10-23

0.14.7.2 Typedef Documentation

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

0.14.7.2.2 Classes `using smtrat::Classes = typedef BacktrackableClasses`

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

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

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

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

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

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

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

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

0.14.7.2.11 EvalDoubleIntervalMap `using smtrat::EvalDoubleIntervalMap = typedef std::map<carl↩ ::Variable, DoubleInterval>`

0.14.7.2.12 EvalRationalIntervalMap `using smtrat::EvalRationalIntervalMap = typedef std::map<carl↩ ::Variable, RationalInterval>`

0.14.7.2.13 Expression `typedef expression::Expression smtrat::Expression`

0.14.7.2.14 Expressions `typedef expression::Expressions smtrat::Expressions`

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

0.14.7.2.16 Formula_Origins `typedef std::map< FormulaT, std::shared_ptr< std::vector<FormulaT> > > smtrat::Formula_Origins`

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

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

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

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

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

0.14.7.2.22 InfinityValue `using smtrat::InfinityValue = typedef carl::InfinityValue`

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

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

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

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

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

0.14.7.2.28 ModelVariable `using smtrat::ModelVariable = typedef carl::ModelVariable`

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

0.14.7.2.30 MonomialMapIterator `typedef std::unordered_map<carl::Variable, carl::Monomial::↩ Arg>::iterator smtrat::MonomialMapIterator`

0.14.7.2.31 MultivariateRootT `using smtrat::MultivariateRootT = typedef carl::Multivariate↩ Root<Poly>`

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

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

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

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

0.14.7.2.36 Rational `using smtrat::Rational = typedef mpq_class`

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

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

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

0.14.7.2.40 SqrtEx `using smtrat::SqrtEx = typedef carl::SqrtEx<Poly>`

0.14.7.2.41 StaticClasses `using smtrat::StaticClasses = typedef EQClasses< StaticUnionFind >`

0.14.7.2.42 Statistics `using smtrat::Statistics = typedef carl::statistics::Statistics`

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

0.14.7.2.44 thread_priority `using smtrat::thread_priority = typedef std::pair<std::size_t, std↩::size_t>`

0.14.7.2.45 Type `using smtrat::Type = typedef carl::UVariable`

0.14.7.2.46 VariableAssignmentT `using smtrat::VariableAssignmentT = typedef carl::Variable↩Assignment<Poly>`

0.14.7.2.47 VariableComparisonT `using smtrat::VariableComparisonT = typedef carl::Variable↩Comparison<Poly>`

0.14.7.2.48 VariableUpperLower

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

0.14.7.3 Enumeration Type Documentation

0.14.7.3.1 Answer

`enum smtrat::Answer`

An enum with the possible answers a Module can give.

Enumerator

SAT	
UNSAT	
UNKNOWN	
ABORTED	
OPTIMAL	

0.14.7.3.2 CoveringStatus

`enum smtrat::CoveringStatus`

Enumerator

partial	
full	
unknown	
failed	

0.14.7.3.3 IsSampleOutsideAlgorithm

`enum smtrat::IsSampleOutsideAlgorithm`

Enumerator

DEFAULT	

0.14.7.3.4 LemmaLevel

`enum smtrat::LemmaLevel`

Enumerator

NONE	
NORMAL	
ADVANCED	

0.14.7.3.5 MaxSMTStrategy

`enum smtrat::MaxSMTStrategy [strong]`

Enumerator

FU_MALIK_INCREMENTAL	
MSU3	
LINEAR_SEARCH	

0.14.7.3.6 SamplingAlgorithm `enum smtrat::SamplingAlgorithm`

Enumerator

LOWER_UPPER_BETWEEN_SAMPLING	

0.14.7.3.7 TheoryGuidedDecisionHeuristicLevel `enum smtrat::TheoryGuidedDecisionHeuristicLevel : unsigned [strong]`

Enumerator

CONFLICT_FIRST	
SATISFIED_FIRST	
DISABLED	

0.14.7.3.8 UnsatCoreStrategy `enum smtrat::UnsatCoreStrategy [strong]`

Enumerator

ModelExclusion	

0.14.7.4 Function Documentation

0.14.7.4.1 abEqualcCheck()

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

0.14.7.4.2 abGreatercCheck()

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

0.14.7.4.3 abLesscCheck()

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

0.14.7.4.4 analyze_file()

```
int smtrat::analyze_file (
    const std::string & )
```

0.14.7.4.5 analyze_formula()

```
analyzer::AnalyzerStatistics & smtrat::analyze_formula (
    const FormulaT & f )
```

0.14.7.4.6 annotateFormula()

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

0.14.7.4.7 CMakeOptions()

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

0.14.7.4.8 convert_to_cnf_dimacs()

```
int smtrat::convert_to_cnf_dimacs (
            const std::string & ,
            const std::string &  )
```

0.14.7.4.9 convert_to_cnf_smtlib()

```
int smtrat::convert_to_cnf_smtlib (
            const std::string & ,
            const std::string &  )
```

0.14.7.4.10 createAbsoluteValuedModel()

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

0.14.7.4.11 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

0.14.7.4.12 createCongruence()

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

0.14.7.4.13 createEquivalentToOriginalMonotonicityOne()

```
FormulaT smtrat::createEquivalentTo↩
OriginalMonotonicityOne (
            smtrat::VariableCapsule variableCapsule,
            smtrat::VariableCapsule variableCapsuleInner )
```

0.14.7.4.14 createEquivalentToOriginalMonotonicityThree()

```
FormulaT smtrat::createEquivalentTo↩
OriginalMonotonicityThree (
            smtrat::VariableCapsule variableCapsule,
            smtrat::VariableCapsule variableCapsuleInner )
```

0.14.7.4.15 createEquivalentToOriginalMonotonicityTwo()

```
FormulaT smtrat::createEquivalentTo↩
OriginalMonotonicityTwo (
    smtrat::VariableCapsule variableCapsule,
    smtrat::VariableCapsule variableCapsuleInner )
```

0.14.7.4.16 createICPGreater()

```
FormulasT smtrat::createICPGreater (
    VariableCapsule variableCapsule,
    RationalCapsule rationalCapsule )
```

0.14.7.4.17 createICPGreaterOne()

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

0.14.7.4.18 createICPGreaterTwo()

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

0.14.7.4.19 createICPLess()

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

0.14.7.4.20 createMonotonicity()

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

0.14.7.4.21 createOriginalMonotonicityOne()

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

0.14.7.4.22 createOriginalMonotonicityThree()

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

0.14.7.4.23 createOriginalMonotonicityTwo()

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

0.14.7.4.24 createTangentPlaneEQ()

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

0.14.7.4.25 createTangentPlaneEQOne()

```
FormulaT smtrat::createTangentPlaneEQOne (
    carl::Variable xVariable,
    carl::Variable yVariable,
```

```
carl::Variable zVariable,
Rational aRational )
```

0.14.7.4.26 createTangentPlaneEQTwo()

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

0.14.7.4.27 createTangentPlaneNEQ()

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

0.14.7.4.28 createTangentPlaneNEQFour()

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

0.14.7.4.29 createTangentPlaneNEQOne()

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

0.14.7.4.30 createTangentPlaneNEQThree()

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

0.14.7.4.31 createTangentPlaneNEQTwo()

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

0.14.7.4.32 createZero()

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

0.14.7.4.33 createZeroOne()

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

0.14.7.4.34 createZeroThree()

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

0.14.7.4.35 createZeroTwo()

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

0.14.7.4.36 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.

0.14.7.4.37 extractRationalCapsule()

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

0.14.7.4.38 extractVariables()

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

0.14.7.4.39 factorial() [1/2]

```
Rational smtrat::factorial (
    Rational n )
```

0.14.7.4.40 factorial() [2/2]

```
Rational smtrat::factorial (
    std::size_t n )
```

0.14.7.4.41 generateAbcPrimeForICP()

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

0.14.7.4.42 generateAbsFormula() [1/2]

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

| x1 | = (and (=> (x1 >= 0) (y1 = x1)) (=> (x1 < 0) (y1 = -x1)))

Parameters

variable	

Returns

0.14.7.4.43 generateAbsFormula() [2/2]

```
FormulaT smtrat::generateAbsFormula (
    carl::Variable variableLeft,
    carl::Variable variableRight,
    carl::Relation relation )
```

| x1 | <= | x1 | = (and (=> (x1 >= 0) (y1 = x1)) (=> (x1 < 0) (y1 = -x1)) (=> (x2 >= 0) (y2 = x2)) (=> (x2 < 0) (y2 = -x2)) (y1 <= y2))

Parameters

variableLeft	
variableRight	
relation	

Returns

0.14.7.4.44 get_name() [1/2]

```
std::string smtrat::get_name (
    IsSampleOutsideAlgorithm samplingAlgorithm ) [inline]
```

0.14.7.4.45 get_name() [2/2]

```
std::string smtrat::get_name (
    SamplingAlgorithm samplingAlgorithm ) [inline]
```

0.14.7.4.46 handle_basic_options()

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

0.14.7.4.47 is_sat()

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

0.14.7.4.48 isAnyRationalIsZero()

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

0.14.7.4.49 isCondition()

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

0.14.7.4.50 operator<<() [1/10]

```
std::ostream& smtrat::operator<< (
    std::ostream & os,
    Answer a ) [inline]
```

0.14.7.4.51 operator<<() [2/10]

```
std::ostream & smtrat::operator<< (
    std::ostream & os,
    CMakeOptionPrinter cmop )
```

0.14.7.4.52 operator<<() [3/10]

```
std::ostream& smtrat::operator<< (
    std::ostream & os,
    const CoveringStatus & status ) [inline]
```

0.14.7.4.53 operator<<() [4/10]

```
template<typename Settings >
std::ostream& smtrat::operator<< (
    std::ostream & os,
    const LevelWiseInformation< Settings > & levelWiseInformation ) [inline]
```

0.14.7.4.54 operator<<() [5/10]

```
template<typename Module >
std::ostream& smtrat::operator<< (
    std::ostream & os,
    const ModuleFactory< Module > & mf ) [inline]
```

0.14.7.4.55 operator<<() [6/10]

```
std::ostream& smtrat::operator<< (
    std::ostream & os,
    const ModuleInput & mi ) [inline]
```

0.14.7.4.56 operator<<() [7/10]

```
template<typename TT , typename C >
std::ostream& smtrat::operator<< (
    std::ostream & os,
    const PriorityQueue< TT, C > & pq )
```

0.14.7.4.57 operator<<() [8/10]

```
std::ostream& smtrat::operator<< (
    std::ostream & os,
    MaxSMTStrategy ucs ) [inline]
```

0.14.7.4.58 operator<<() [9/10]

```
std::ostream& smtrat::operator<< (
    std::ostream & os,
    Module::LemmaType lt ) [inline]
```

0.14.7.4.59 operator<<() [10/10]

```
std::ostream& smtrat::operator<< (
    std::ostream & os,
    UnsatCoreStrategy ucs ) [inline]
```

0.14.7.4.60 parse_formula()

```
FormulaT smtrat::parse_formula (
    const std::string & filename )
```

Loads the smt2 file specified in filename and returns the formula.
This function ignores most SMT-LIB commands but simply accumulated all asserted formulas.

0.14.7.4.61 parse_smtlib()

```
FormulaT smtrat::parse_smtlib (
    const std::string & )
```

0.14.7.4.62 parseInput()

```
template<typename Executor >
bool smtrat::parseInput (
    const std::string & pathToInputFile,
```

```
Executor & e,
bool & queueInstructions )
```

0.14.7.4.63 parseSMT2File()

```
bool smtrat::parseSMT2File (
    parser::InstructionHandler & handler,
    bool queueInstructions,
    std::istream & input )
```

0.14.7.4.64 preprocess_file()

```
int smtrat::preprocess_file (
    const std::string & filename,
    const std::string & outfile )
```

Loads the smt2 file specified in filename and runs the preprocessing strategy.
The resulting (simplified) problem is written to outfile (or stdout if outfile is empty).

0.14.7.4.65 run_dimacs_file()

```
template<typename Strategy >
int smtrat::run_dimacs_file (
    Strategy & ,
    const std::string &  )
```

0.14.7.4.66 run_opb_file()

```
template<typename Strategy >
int smtrat::run_opb_file (
    Strategy & ,
    const std::string &  )
```

0.14.7.4.67 Settings()

```
const settings::Settings& smtrat::Settings ( ) [inline]
```

0.14.7.4.68 settings_analyzer()

```
const auto& smtrat::settings_analyzer ( ) [inline]
```

0.14.7.4.69 settings_core()

```
const auto& smtrat::settings_core ( ) [inline]
```

0.14.7.4.70 settings_module()

```
const auto& smtrat::settings_module ( ) [inline]
```

0.14.7.4.71 settings_parser()

```
const auto& smtrat::settings_parser ( ) [inline]
```

0.14.7.4.72 settings_solver()

```
const auto& smtrat::settings_solver ( ) [inline]
```

0.14.7.4.73 settings_statistics()

```
const auto& smtrat::settings_statistics ( ) [inline]
```

0.14.7.4.74 settings_validation()

```
const auto& smtrat::settings_validation ( ) [inline]
```

0.14.7.4.75 statistics_get()

```
template<typename T >
auto& smtrat::statistics_get (
    const std::string & name )
```

0.14.7.4.76 testExpression() `void smtrat::testExpression ( )`

0.14.7.4.77 validation_get() `auto& smtrat::validation_get (`
`const std::string & channel,`
`const std::string & file,`
`int line ) [inline]`

0.14.7.4.78 visiting() `void smtrat::visiting (`
`const Expression & expr )`

0.14.7.5 Variable Documentation

0.14.7.5.1 COMMANDER_GROUP_SIZE `constexpr int smtrat::COMMANDER_GROUP_SIZE = 4 [static], [constexpr]`

0.14.7.5.2 EMPTY_MODEL `const Model smtrat::EMPTY_MODEL = Model() [static]`

0.14.8 smtrat::analyzer Namespace Reference

Data Structures

- struct DegreeCollector
- struct AnalysisSettings
- struct AnalyzerStatistics

Functions

- template<typename P >
  void collect_projection_size (const std::string &prefix, const P &projection, AnalyzerStatistics &stats)
- template<typename Settings >
  void perform_projection (const std::string &prefix, const std::set< ConstraintT > &constraints, AnalyzerStatistics &stats)
- void analyze_cad_projections (const FormulaT &f, AnalyzerStatistics &stats)
- void analyze_cnf (const FormulaT &f, AnalyzerStatistics &stats)
- void analyze_formula_types (const FormulaT &f, AnalyzerStatistics &stats)
- void analyze_variables (const FormulaT &f, AnalyzerStatistics &stats)
- template<typename T >
  void registerAnalyzerSettings (T &parser)

0.14.8.1 Function Documentation

0.14.8.1.1 analyze_cad_projections() `void smtrat::analyzer::analyze_cad_projections (`
`const FormulaT & f,`
`AnalyzerStatistics & stats )`

0.14.8.1.2 analyze_cnf() `void smtrat::analyzer::analyze_cnf (`
`const FormulaT & f,`
`AnalyzerStatistics & stats )`

0.14.8.1.3 analyze_formula_types()

```
void smtrat::analyzer::analyze_formula_types (
    const FormulaT & f,
    AnalyzerStatistics & stats )
```

0.14.8.1.4 analyze_variables()

```
void smtrat::analyzer::analyze_variables (
    const FormulaT & f,
    AnalyzerStatistics & stats )
```

0.14.8.1.5 collect_projection_size()

```
template<typename P >
void smtrat::analyzer::collect_projection_size (
    const std::string & prefix,
    const P & projection,
    AnalyzerStatistics & stats )
```

0.14.8.1.6 perform_projection()

```
template<typename Settings >
void smtrat::analyzer::perform_projection (
    const std::string & prefix,
    const std::set< ConstraintT > & constraints,
    AnalyzerStatistics & stats )
```

0.14.8.1.7 registerAnalyzerSettings()

```
template<typename T >
void smtrat::analyzer::registerAnalyzerSettings (
    T & parser )
```

0.14.9 smtrat::cad Namespace Reference

Namespaces

- debug
- full
- full_ec
- preprocessor
- projection
- projection_compare
- sample_compare

  Contains comparison operators for samples and associated helpers.
- variable_ordering

Data Structures

- class CAD
- class LiftingTree
- class Sample
- struct FullSampleComparator
- struct FullSampleComparator< Iterator, FullSampleCompareStrategy::Value >
- struct FullSampleComparator< Iterator, FullSampleCompareStrategy::Type >
- struct FullSampleComparator< Iterator, FullSampleCompareStrategy::T >
- class SampleIteratorQueue
- class BaseProjection
- struct PolynomialComparator
- class PolynomialLiftingQueue
- class Projection

- class ModelBasedProjection
- class Projection< Incrementality::FULL, Backtracking::HIDE, Settings >
- class Projection< Incrementality::FULL, BT, Settings >
- class ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >

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

  This class implements a projection that supports no incrementality and expects backtracking to be in order.
- class Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >

  This class implements a projection that supports no incrementality and allows backtracking to be out of order.
- class Projection< Incrementality::SIMPLE, BT, Settings >
- class ProjectionGlobalInformation
- class ProjectionLevelInformation
- class ProjectionPolynomialInformation
- class ProjectionInformation
- struct ProjectionOperator
- struct BaseSettings
- class CADConstraints
- struct CADCore
- struct CADCore< CoreHeuristic::BySample >
- struct CADCore< CoreHeuristic::PreferProjection >
- struct CADCore< CoreHeuristic::PreferSampling >
- struct CADCore< CoreHeuristic::Interleave >
- struct CADCore< CoreHeuristic::EnumerateAll >
- struct CADPreprocessorSettings
- class CADPreprocessor
- class ConflictGraph

  Representation of a bipartite graph (C+S, E) of vertices C and S, representing the constraints and samples, respectively.
- class MISGeneration
- class Origin

  This class represents one or more origins of some object.
- struct PreprocessorSettings
- class Preprocessor

Typedefs

- using RAN = smtrat::RAN
- using ConstraintSelection = carl::Bitset
- using OptionalID = std::optional< std::size_t >
- using Assignment = std::map< carl::Variable, RAN >
- using SampleLiftedWith = carl::Bitset
- using SampleRootOf = carl::Bitset
- using UPoly = carl::UnivariatePolynomial< Poly >
- template<typename Settings >
  using ProjectionT = Projection< Settings::incrementality, Settings::backtracking, Settings >
- template<typename Settings >
  using ModelBasedProjectionT = ModelBasedProjection< Settings::incrementality, Settings::backtracking, Settings >

Enumerations

- enum class Incrementality { NONE , SIMPLE , FULL }
- enum class Backtracking { ORDERED , UNORDERED , HIDE }
- enum class ProjectionType {
  Collins , Hong , McCallum , McCallum_partial ,
  Lazard , Brown }
- enum class ProjectionCompareStrategy {
  D , PD , SD , ID ,
  LD , Default = ID }
- enum class SampleCompareStrategy {
  T , TLSA , TSA , TS ,
  LT , LTA , LTS , LTSA ,
  LS , S , Type , Value ,
  Default = LT }
- enum class FullSampleCompareStrategy { Type , Value , T , Default = T }
- enum class MISHeuristic {
  TRIVIAL , GREEDY , GREEDY_PRE , GREEDY_WEIGHTED ,
  EXACT , HYBRID }
- enum class CoreHeuristic {
  BySample , PreferProjection , PreferSampling , EnumerateAll ,
  Interleave }

Functions

- template<typename I , typename C >
  std::ostream & operator<< (std::ostream &os, const SampleIteratorQueue< I, C > &siq)
- template<typename PG >
  std::ostream & operator<< (std::ostream &os, const PolynomialLiftingQueue< PG > &plq)
- template<typename S >
  std::ostream & operator<< (std::ostream &os, const Projection< Incrementality::FULL, Backtracking::HIDE, S > &p)
- std::ostream & operator<< (std::ostream &os, const full::Polynomial &p)
- template<typename S , Backtracking B>
  std::ostream & operator<< (std::ostream &os, const Projection< Incrementality::FULL, B, S > &p)
- template<typename S >
  std::ostream & operator<< (std::ostream &os, const ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, S > &p)
- template<typename S >
  std::ostream & operator<< (std::ostream &os, const Projection< Incrementality::NONE, Backtracking::ORDERED, S > &p)
- template<typename S >
  std::ostream & operator<< (std::ostream &os, const Projection< Incrementality::NONE, Backtracking::UNORDERED, S > &p)
- template<typename S , Backtracking B>
  std::ostream & operator<< (std::ostream &os, const Projection< Incrementality::SIMPLE, B, S > &p)
- template<typename Poly >
  projection_compare::Candidate< Poly > candidate (const Poly &p, const Poly &q, std::size_t level)
- std::ostream & operator<< (std::ostream &os, const ProjectionGlobalInformation &gi)
- std::ostream & operator<< (std::ostream &os, const ProjectionLevelInformation::LevelInfo &li)
- std::ostream & operator<< (std::ostream &os, const ProjectionPolynomialInformation::PolyInfo &pi)
- template<Backtracking BT>
  std::ostream & operator<< (std::ostream &os, const CADConstraints< BT > &cc)
- const auto & settings_cadpp ()
- std::ostream & operator<< (std::ostream &os, const CADPreprocessor &cadpp)
- std::ostream & operator<< (std::ostream &os, const ConflictGraph &cg)
- std::ostream & operator<< (std::ostream &os, const Preprocessor &cadpp)

0.14.9.1 Typedef Documentation

0.14.9.1.1 Assignment

```
using smtrat::cad::Assignment = typedef std::map<carl::Variable, RAN>
```

0.14.9.1.2 ConstraintSelection

```
using smtrat::cad::ConstraintSelection = typedef carl::Bitset
```

0.14.9.1.3 ModelBasedProjectionT

```
template<typename Settings >
using smtrat::cad::ModelBasedProjectionT = typedef ModelBasedProjection<Settings::incrementality,
Settings::backtracking, Settings>
```

0.14.9.1.4 OptionalID

```
using smtrat::cad::OptionalID = typedef std::optional<std::size_t>
```

0.14.9.1.5 ProjectionT

```
template<typename Settings >
using smtrat::cad::ProjectionT = typedef Projection<Settings::incrementality, Settings::backtracking,
Settings>
```

0.14.9.1.6 RAN

```
using smtrat::cad::RAN = typedef smtrat::RAN
```

0.14.9.1.7 SampleLiftedWith

```
using smtrat::cad::SampleLiftedWith = typedef carl::Bitset
```

0.14.9.1.8 SampleRootOf

```
using smtrat::cad::SampleRootOf = typedef carl::Bitset
```

0.14.9.1.9 UPoly

```
using smtrat::cad::UPoly = typedef carl::UnivariatePolynomial<Poly>
```

0.14.9.2 Enumeration Type Documentation

0.14.9.2.1 Backtracking

```
enum smtrat::cad::Backtracking [strong]
```

Enumerator

ORDERED	
UNORDERED	
HIDE	

0.14.9.2.2 CoreHeuristic

```
enum smtrat::cad::CoreHeuristic [strong]
```

Enumerator

BySample	
PreferProjection	
PreferSampling	

Enumerator

EnumerateAll	
Interleave	

0.14.9.2.3 FullSampleCompareStrategy `enum smtrat::cad::FullSampleCompareStrategy [strong]`

Enumerator

Type	
Value	
T	
Default	

0.14.9.2.4 Incrementality `enum smtrat::cad::Incrementality [strong]`

Enumerator

NONE	
SIMPLE	
FULL	

0.14.9.2.5 MISHeuristic `enum smtrat::cad::MISHeuristic [strong]`

Enumerator

TRIVIAL	
GREEDY	
GREEDY_PRE	
GREEDY_WEIGHTED	
EXACT	
HYBRID	

0.14.9.2.6 ProjectionCompareStrategy `enum smtrat::cad::ProjectionCompareStrategy [strong]`

Enumerator

D	
PD	
SD	
ID	
LD	
Default	

0.14.9.2.7 ProjectionType `enum smtrat::cad::ProjectionType [strong]`

Enumerator

Collins	
Hong	
McCallum	
McCallum_partial	
Lazard	
Brown	

0.14.9.2.8 SampleCompareStrategy `enum smtrat::cad::SampleCompareStrategy [strong]`

Enumerator

T	
TLSA	
TSA	
TS	
LT	
LTA	
LTS	
LTSA	
LS	
S	
Type	
Value	
Default	

0.14.9.3 Function Documentation

0.14.9.3.1 candidate()

```
template<typename Poly >
projection_compare::Candidate<Poly> smtrat::cad::candidate (
            const Poly & p,
            const Poly & q,
            std::size_t level )
```

0.14.9.3.2 operator<<() [1/16]

```
template<Backtracking BT>
std::ostream& smtrat::cad::operator<< (
            std::ostream & os,
            const CADConstraints< BT > & cc )
```

0.14.9.3.3 operator<<() [2/16]

```
std::ostream& smtrat::cad::operator<< (
            std::ostream & os,
            const CADPreprocessor & cadpp ) [inline]
```

0.14.9.3.4 operator<<() [3/16]

```
std::ostream & smtrat::cad::operator<< (
            std::ostream & os,
            const ConflictGraph & cg )
```

0.14.9.3.5 operator<<() [4/16]

```
std::ostream& smtrat::cad::operator<< (
            std::ostream & os,
            const full::Polynomial & p )  [inline]
```

0.14.9.3.6 operator<<() [5/16]

```
template<typename S >
std::ostream& smtrat::cad::operator<< (
            std::ostream & os,
            const ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, S > & p
)
```

0.14.9.3.7 operator<<() [6/16]

```
template<typename PG >
std::ostream& smtrat::cad::operator<< (
            std::ostream & os,
            const PolynomialLiftingQueue< PG > & plq )  [inline]
```

0.14.9.3.8 operator<<() [7/16]

```
std::ostream& smtrat::cad::operator<< (
            std::ostream & os,
            const Preprocessor & cadpp )  [inline]
```

0.14.9.3.9 operator<<() [8/16]

```
template<typename S , Backtracking B>
std::ostream& smtrat::cad::operator<< (
            std::ostream & os,
            const Projection< Incrementality::FULL, B, S > & p )
```

0.14.9.3.10 operator<<() [9/16]

```
template<typename S >
std::ostream& smtrat::cad::operator<< (
            std::ostream & os,
            const Projection< Incrementality::FULL, Backtracking::HIDE, S > & p )
```

0.14.9.3.11 operator<<() [10/16]

```
template<typename S >
std::ostream& smtrat::cad::operator<< (
            std::ostream & os,
            const Projection< Incrementality::NONE, Backtracking::ORDERED, S > & p )
```

0.14.9.3.12 operator<<() [11/16]

```
template<typename S >
std::ostream& smtrat::cad::operator<< (
            std::ostream & os,
            const Projection< Incrementality::NONE, Backtracking::UNORDERED, S > & p )
```

0.14.9.3.13 operator<<() [12/16]

```
template<typename S , Backtracking B>
std::ostream& smtrat::cad::operator<< (
            std::ostream & os,
            const Projection< Incrementality::SIMPLE, B, S > & p )
```

0.14.9.3.14 operator<<() [13/16]

```
std::ostream& smtrat::cad::operator<< (
    std::ostream & os,
    const ProjectionGlobalInformation & gi ) [inline]
```

0.14.9.3.15 operator<<() [14/16]

```
std::ostream& smtrat::cad::operator<< (
    std::ostream & os,
    const ProjectionLevelInformation::LevelInfo & li ) [inline]
```

0.14.9.3.16 operator<<() [15/16]

```
std::ostream& smtrat::cad::operator<< (
    std::ostream & os,
    const ProjectionPolynomialInformation::PolyInfo & pi ) [inline]
```

0.14.9.3.17 operator<<() [16/16]

```
template<typename I , typename C >
std::ostream& smtrat::cad::operator<< (
    std::ostream & os,
    const SampleIteratorQueue< I, C > & siq ) [inline]
```

0.14.9.3.18 settings_cadpp()

```
const auto & smtrat::cad::settings_cadpp ( ) [inline]
```

0.14.10 smtrat::cad::debug Namespace Reference

Data Structures

- struct DotSubgraph
- struct IDSanitizer
- struct UnifiedData
- class TikzBasePrinter
- class TikzDAGPrinter
- class TikzTreePrinter
- class TikzHistoryPrinter

0.14.11 smtrat::cad::full Namespace Reference

Typedefs

- using Polynomial = std::optional< std::pair< UPoly, Origin > >

0.14.11.1 Typedef Documentation

0.14.11.1.1 Polynomial

```
using smtrat::cad::full::Polynomial = typedef std::optional<std::pair<UPoly,Origin>
>
```

0.14.12 smtrat::cad::full_ec Namespace Reference

Typedefs

- using Polynomial = std::optional< UPoly >

0.14.12.1 Typedef Documentation

0.14.12.1.1 Polynomial `using smtrat::cad::full_ec::Polynomial = typedef std::optional<UPoly>`

0.14.13 smtrat::cad::preprocessor Namespace Reference

Data Structures

- struct Origins
- class AssignmentCollector
- class ResultantRule
- struct ConstraintUpdate

Functions

- std::size_t complexity (const std::vector< FormulaT > &origin)

0.14.13.1 Function Documentation

0.14.13.1.1 complexity() `std::size_t smtrat::cad::preprocessor::complexity ( const std::vector< FormulaT > & origin ) [inline]`

0.14.14 smtrat::cad::projection Namespace Reference

Namespaces

- brown

  Contains the implementation of Browns projection operator as specified in **?** *after Theorem 3.1.*
- collins

  Contains the implementation of Collins projection operator as specified in **?** *below Theorem 4.*
- hong

  Contains the implementation of Hongs projection operator as specified in **?** *in Section 2.2.*
- lazard

  Contains the implementation of Lazards projection operator as specified in ***[Lauard1994]****.*
- mccallum

  Contains the implementation of McCallums projection operator as specified in **?***.*
- mccallum_partial

  Contains the implementation of an optimized version McCallums projection operator.

Data Structures

- struct Reducta

  Construct the set of reducta of the given polynomial.

Functions

- bool canBeRemoved (const UPoly &p)

  Checks whether a polynomial can safely be ignored.
- bool canBeForwarded (std::size_t, const UPoly &p)

  Checks whether a polynomial can safely be forwarded to the next level.
- template<typename Poly >
  bool doesNotVanish (const Poly &p)

Tries to determine whether the given Poly vanishes for some assignment.

- template<typename Poly >
  Poly normalize (const Poly &p)

  Normalizes the given Poly by removing constant and duplicate factors.
- template<typename Poly >
  Poly resultant (carl::Variable variable, const Poly &p, const Poly &q)

  Computes the resultant of two polynomials.
- template<typename Poly >
  Poly discriminant (carl::Variable variable, const Poly &p)

  Computes the discriminant of a polynomial.
- template<typename Poly >
  std::vector< Poly > PSC (const Poly &p, const Poly &q)

  Computes the Principal Subresultant Coefficients of two polynomials.
- template<typename Poly , typename Callback >
  void returnPoly (const Poly &p, Callback &&cb)

0.14.14.1 Function Documentation

0.14.14.1.1 canBeForwarded()

```
bool smtrat::cad::projection::canBeForwarded (
    std::size_t ,
    const UPoly & p ) [inline]
```

Checks whether a polynomial can safely be forwarded to the next level.

0.14.14.1.2 canBeRemoved()

```
bool smtrat::cad::projection::canBeRemoved (
    const UPoly & p ) [inline]
```

Checks whether a polynomial can safely be ignored.

0.14.14.1.3 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.

0.14.14.1.4 doesNotVanish()

```
template<typename Poly >
bool smtrat::cad::projection::doesNotVanish (
    const Poly & p )
```

Tries to determine whether the given Poly vanishes for some assignment.
Returns true if the Poly is guaranteed not to vanish. Returns false otherwise.

0.14.14.1.5 normalize()

```
template<typename Poly >
Poly smtrat::cad::projection::normalize (
            const Poly & p )
```

Normalizes the given Poly by removing constant and duplicate factors.

0.14.14.1.6 PSC()

```
template<typename Poly >
std::vector<Poly> smtrat::cad::projection::PSC (
            const Poly & p,
            const Poly & q )
```

Computes the Principal Subresultant Coefficients of two polynomials.

0.14.14.1.7 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.

0.14.14.1.8 returnPoly()

```
template<typename Poly , typename Callback >
void smtrat::cad::projection::returnPoly (
            const Poly & p,
            Callback && cb )
```

0.14.15 smtrat::cad::projection::brown Namespace Reference

Contains the implementation of Browns projection operator as specified in **?** after Theorem 3.1.

Functions

- template<typename Poly , typename Callback >
  void single (const Poly &p, carl::Variable variable, Callback &&cb)

  Implements the part of Browns projection operator from **?** *that deals with a single polynomial* $p \leftarrow : \{leading_coeff(p), discriminant(p)\}$.
- template<typename Poly , typename Callback >
  void paired (const Poly &p, const UPoly &q, carl::Variable variable, Callback &&cb)

  Implements the part of Browns projection operator from **?** *that deals with two polynomials* p *and* q *which is just the respective part of McCallums projection operator mccallum::paired.*

0.14.15.1 Detailed Description

Contains the implementation of Browns projection operator as specified in **?** after Theorem 3.1.

0.14.15.2 Function Documentation

0.14.15.2.1 paired()

```
template<typename Poly , typename Callback >
void smtrat::cad::projection::brown::paired (
            const Poly & p,
            const UPoly & q,
            carl::Variable variable,
            Callback && cb )
```

Implements the part of Browns projection operator from **?** that deals with two polynomials p and q which is just the respective part of McCallums projection operator mccallum::paired.

0.14.15.2.2 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 $\mathrm{p}\hookleftarrow$: $\{leading_c oeff(p), discriminant(p)\}$.

0.14.16 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* $q\hookleftarrow$ *:* $\bigcup_{B_1\in RED(p),B_2\in RED(q)} PSC(B_1,B_2)$.

0.14.16.1 Detailed Description

Contains the implementation of Collins projection operator as specified in **?** below Theorem 4.

0.14.16.2 Function Documentation

0.14.16.2.1 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 $\mathrm{q}\hookleftarrow$: $\bigcup_{B_1\in RED(p),B_2\in RED(q)} PSC(B_1,B_2)$.

0.14.16.2.2 single()

```
template<typename Poly , typename Callback >
void smtrat::cad::projection::collins::single (
            const Poly & p,
            carl::Variable variable,
            Callback && cb )
```

Implements the part of Collins projection operator from **?** that deals with a single polynomial $p \leftarrow : \bigcup_{B \in RED(p)} \{ldcf(B)\} \cup PSC(B, B')$.

0.14.17 smtrat::cad::projection::hong Namespace Reference

Contains the implementation of Hongs projection operator as specified in **?** in Section 2.2.

Functions

- template<typename Poly , typename Callback >
  void single (const Poly &p, carl::Variable variable, Callback &&cb)

  Implements the part of Hongs projection operator from **?** *that deals with a single polynomial p which is just the respective part of Collins' projection operator collins::single.*
- template<typename Poly , typename Callback >
  void paired (const Poly &p, const UPoly &q, carl::Variable variable, Callback &&cb)

  Implements the part of Hongs projection operator from **?** *that deals with two polynomials p and $q \leftarrow : \bigcup_{F \in RED(p)} PSC(F, q)$.*

0.14.17.1 Detailed Description

Contains the implementation of Hongs projection operator as specified in **?** in Section 2.2.

0.14.17.2 Function Documentation

0.14.17.2.1 paired()

```
template<typename Poly , typename Callback >
void smtrat::cad::projection::hong::paired (
            const Poly & p,
            const UPoly & q,
            carl::Variable variable,
            Callback && cb )
```

Implements the part of Hongs projection operator from **?** that deals with two polynomials p and $q \leftarrow : \bigcup_{F \in RED(p)} PSC(F, q)$.

0.14.17.2.2 single()

```
template<typename Poly , typename Callback >
void smtrat::cad::projection::hong::single (
            const Poly & p,
            carl::Variable variable,
            Callback && cb )
```

Implements the part of Hongs projection operator from **?** that deals with a single polynomial p which is just the respective part of Collins' projection operator collins::single.

0.14.18 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* $p \hookleftarrow : \{leading_c oeff(p), trailing_c oeff(p), discriminant(p)\}$.

- 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.*

0.14.18.1 Detailed Description

Contains the implementation of Lazards projection operator as specified in **[Lauard1994]**.

0.14.18.2 Function Documentation

0.14.18.2.1 paired()

```
template<typename Poly , typename Callback >
void smtrat::cad::projection::lazard::paired (
            const Poly & p,
            const UPoly & q,
            carl::Variable variable,
            Callback && cb )
```

Implements the part of Lazard projection operator from **?** that deals with two polynomials p and q which is just the respective part of McCallums projection operator mccallum::paired.

0.14.18.2.2 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 $\mathrm{p} \hookleftarrow : \{leading_c oeff(p), trailing_c oeff(p), discriminant(p)\}$.

0.14.19 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* $q \hookleftarrow : \{resultant(p, q)\}$.

0.14.19.1 Detailed Description

Contains the implementation of McCallums projection operator as specified in **?**.

0.14.19.2 Function Documentation

0.14.19.2.1 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↩ : $\{resultant(p, q)\}$.

0.14.19.2.2 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↩ : $coefficients(p) \cup \{discriminant(p)\}$.

0.14.20 smtrat::cad::projection::mccallum_partial Namespace Reference

Contains the implementation of an optimized version McCallums projection operator.

Functions

- template<typename Poly , typename Callback >
  void single (const Poly &p, carl::Variable variable, Callback &&cb)
- template<typename Poly , typename Callback >
  void paired (const Poly &p, const UPoly &q, carl::Variable variable, Callback &&cb)

0.14.20.1 Detailed Description

Contains the implementation of an optimized version McCallums projection operator.
It is based on the observation by Brown that if some coefficient does not vanish only the leading coefficient is required.

0.14.20.2 Function Documentation

0.14.20.2.1 paired()

```
template<typename Poly , typename Callback >
void smtrat::cad::projection::mccallum_partial::paired (
            const Poly & p,
            const UPoly & q,
            carl::Variable variable,
            Callback && cb )
```

0.14.20.2.2 single()

```
template<typename Poly , typename Callback >
void smtrat::cad::projection::mccallum_partial::single (
            const Poly & p,
            carl::Variable variable,
            Callback && cb )
```

0.14.21 smtrat::cad::projection_compare Namespace Reference

Data Structures

- struct level

- struct degree
- struct type
- struct ProjectionComparator_impl
- struct ProjectionComparator
- struct ProjectionComparator< ProjectionCompareStrategy::D >
- struct ProjectionComparator< ProjectionCompareStrategy::PD >
- struct ProjectionComparator< ProjectionCompareStrategy::SD >
- struct ProjectionComparator< ProjectionCompareStrategy::ID >
- struct ProjectionComparator< ProjectionCompareStrategy::LD >

Typedefs

- template<typename Poly >
  using Candidate = std::tuple< const Poly &, const Poly &, std::size_t >
- using lt = std::less<>
- using gt = std::greater<>

Functions

- template<typename Poly >
  std::ostream & operator<< (std::ostream &os, const Candidate< Poly > &c)
- template<typename Poly >
  auto get (const Candidate< Poly > &c, level)
- template<typename Poly >
  auto get (const Candidate< Poly > &c, degree)
- template<typename Poly >
  auto get (const Candidate< Poly > &c, type)
- template<typename Poly , typename tag , typename F >
  int compareCriterion (const Candidate< Poly > &lhs, const Candidate< Poly > &rhs, tag t, F &&f)

  Compares the criterion given by `t` of two samples `lhs` and `rhs` using a comparator `f`.
- template<typename Poly >
  bool compare (const Candidate< Poly > &lhs, const Candidate< Poly > &rhs)
- template<typename Poly , typename tag , typename F , typename... Tail>
  bool compare (const Candidate< Poly > &lhs, const Candidate< Poly > &rhs)

0.14.21.1 Typedef Documentation

0.14.21.1.1 Candidate `template<typename Poly >`

```
using smtrat::cad::projection_compare::Candidate = typedef std::tuple<const Poly&, const Poly&,
std::size_t>
```

0.14.21.1.2 gt `using smtrat::cad::projection_compare::gt = typedef std::greater<>`

0.14.21.1.3 lt `using smtrat::cad::projection_compare::lt = typedef std::less<>`

0.14.21.2 Function Documentation

0.14.21.2.1 compare() [1/2] `template<typename Poly >`

```
bool smtrat::cad::projection_compare::compare (
            const Candidate< Poly > & lhs,
            const Candidate< Poly > & rhs )
```

0.14.21.2.2 compare() [2/2]

```
template<typename Poly , typename tag , typename F , typename...
Tail>
bool smtrat::cad::projection_compare::compare (
            const Candidate< Poly > & lhs,
            const Candidate< Poly > & rhs )
```

0.14.21.2.3 compareCriterion()

```
template<typename Poly , typename tag , typename F >
int smtrat::cad::projection_compare::compareCriterion (
            const Candidate< Poly > & lhs,
            const Candidate< Poly > & rhs,
            tag t,
            F && f )
```

Compares the criterion given by `t` of two samples `lhs` and `rhs` using a comparator `f`.
Returns `0` if the values are the same, `-1` if `lhs` should be used first and `1` if `rhs` should be used first.

0.14.21.2.4 get() [1/3]

```
template<typename Poly >
auto smtrat::cad::projection_compare::get (
            const Candidate< Poly > & c,
            degree  )
```

0.14.21.2.5 get() [2/3]

```
template<typename Poly >
auto smtrat::cad::projection_compare::get (
            const Candidate< Poly > & c,
            level  )
```

0.14.21.2.6 get() [3/3]

```
template<typename Poly >
auto smtrat::cad::projection_compare::get (
            const Candidate< Poly > & c,
            type  )
```

0.14.21.2.7 operator<<()

```
template<typename Poly >
std::ostream& smtrat::cad::projection_compare::operator<< (
            std::ostream & os,
            const Candidate< Poly > & c )
```

0.14.22 smtrat::cad::sample_compare Namespace Reference

Contains comparison operators for samples and associated helpers.

Data Structures

- struct level
- struct size
- struct absvalue
- struct type
- struct SampleComparator_impl
- struct SampleComparator
- struct SampleComparator< Iterator, SampleCompareStrategy::Value >
- struct SampleComparator< Iterator, SampleCompareStrategy::Type >
- struct SampleComparator< Iterator, SampleCompareStrategy::T >
- struct SampleComparator< Iterator, SampleCompareStrategy::TLSA >
- struct SampleComparator< Iterator, SampleCompareStrategy::TSA >

- struct SampleComparator< Iterator, SampleCompareStrategy::TS >
- struct SampleComparator< Iterator, SampleCompareStrategy::LT >
- struct SampleComparator< Iterator, SampleCompareStrategy::LTA >
- struct SampleComparator< Iterator, SampleCompareStrategy::LTS >
- struct SampleComparator< Iterator, SampleCompareStrategy::LTSA >
- struct SampleComparator< Iterator, SampleCompareStrategy::LS >
- struct SampleComparator< Iterator, SampleCompareStrategy::S >

Typedefs

- using lt = std::less<>
- using gt = std::greater<>

Functions

- template<typename It >
  auto get (const It &it, level)
- template<typename It >
  auto get (const It &it, size)
- template<typename It >
  auto get (const It &it, absvalue)
- template<typename It >
  auto get (const It &it, type)
- template<typename It , typename tag , typename F >
  int compareCriterion (const It &lhs, const It &rhs, tag t, F &&f)

  Compares the criterion given by `t` of two samples `lhs` and `rhs` using a comparator `f`.
- template<typename It >
  bool compare (const It &lhs, const It &rhs)
- template<typename It , typename tag , typename F , typename... Tail>
  bool compare (const It &lhs, const It &rhs)

0.14.22.1 Detailed Description

Contains comparison operators for samples and associated helpers.
The comparators implement a `less-than` operators and large samples are considered first. Hence if `lhs` has greater priority than `rhs`, the result should be `false`.

0.14.22.2 Typedef Documentation

0.14.22.2.1 gt `using smtrat::cad::sample_compare::gt = typedef std::greater<>`

0.14.22.2.2 lt `using smtrat::cad::sample_compare::lt = typedef std::less<>`

0.14.22.3 Function Documentation

0.14.22.3.1 compare() [1/2] `template<typename It >`

```
bool smtrat::cad::sample_compare::compare (
            const It & lhs,
            const It & rhs )
```

0.14.22.3.2 compare() [2/2]

```
template<typename It , typename tag , typename F , typename...
Tail>
bool smtrat::cad::sample_compare::compare (
            const It & lhs,
            const It & rhs )
```

0.14.22.3.3 compareCriterion()

```
template<typename It , typename tag , typename F >
int smtrat::cad::sample_compare::compareCriterion (
            const It & lhs,
            const It & rhs,
            tag t,
            F && f )
```

Compares the criterion given by `t` of two samples `lhs` and `rhs` using a comparator `f`.
Returns `0` if the values are the same, `-1` if `lhs` should be used first and `1` if `rhs` should be used first.

0.14.22.3.4 get() [1/4]

```
template<typename It >
auto smtrat::cad::sample_compare::get (
            const It & it,
            absvalue  )
```

0.14.22.3.5 get() [2/4]

```
template<typename It >
auto smtrat::cad::sample_compare::get (
            const It & it,
            level  )
```

0.14.22.3.6 get() [3/4]

```
template<typename It >
auto smtrat::cad::sample_compare::get (
            const It & it,
            size  )
```

0.14.22.3.7 get() [4/4]

```
template<typename It >
auto smtrat::cad::sample_compare::get (
            const It & it,
            type  )
```

0.14.23 smtrat::cad::variable_ordering Namespace Reference

Functions

- std::vector< carl::Variable > triangular_ordering (const std::vector< Poly > &polys)

0.14.23.1 Function Documentation

0.14.23.1.1 triangular_ordering()

```
std::vector< carl::Variable > smtrat::cad::variable_ordering↩
::triangular_ordering (
            const std::vector< Poly > & polys )
```

0.14.24 smtrat::cadcells Namespace Reference

A framework for sample-based CAD algorithms.

Namespaces

- algorithms

  Various algorithms as well as helper functions for developing new algorithms.
- datastructures

  Main datastructures.
- helper
- operators

  Projection operators.
- representation

  Heuristics for computing representations.

Typedefs

- using VariableOrdering = std::vector< carl::Variable >
- using Polynomial = carl::ContextPolynomial< Rational >
- using Constraint = carl::BasicConstraint< Polynomial >
- using MultivariateRoot = carl::MultivariateRoot< Polynomial >
- using VariableComparison = carl::VariableComparison< Polynomial >
- using Atom = std::variant< Constraint, VariableComparison >
- using Disjunction = std::vector< Atom >
- using CNF = std::vector< Disjunction >
- using RAN = Polynomial::RootType
- using Assignment = carl::Assignment< RAN >

Variables

- static const Assignment empty_assignment

0.14.24.1 Detailed Description

A framework for sample-based CAD algorithms.
This is a highly modular framework for sample-based CAD algorithms, i.e. single cell construction and coverings. The basic idea is to have properties (of some polynomials or real root functions); each property has a level w.r.t. to a variable ordering (i.e. the index of the main variable of a polynomial) and rules operating on them, each replacing a property by a "simpler" set of properties (eventually reducing the level). At some stage, we delineate properties (that is, ordering the root under a partial sample), determine an ordering and cell boundaries (called representation) to continue proving properties. For more details on the general framework, we refer to the paper.
The structure of this implementation is as follows:

- cadcells::datastructures contains the main datastructures. Read here for more details on the general structure of the framework.
- cadcells::operators defines the properties, the rules and methods to delineate properties. These are used by operators which provide an interface for performing projections on certain steps.
- cadcells::representation provides heuristics for computing the representations for cells, coverings and delineations.
- cadcells::algorithms contains helper methods, building blocks for algorithms as well as algorithms themselves. Go here for usage of the framework and high-level interfaces.

0.14.24.1.1 Quick start

For an introduction, we refer to the code of algorithms::onecell.

0.14.24.2 Typedef Documentation

0.14.24.2.1 Assignment

`using smtrat::cadcells::Assignment = typedef carl::Assignment<RAN>`

0.14.24.2.2 Atom

`using smtrat::cadcells::Atom = typedef std::variant<Constraint, VariableComparison>`

0.14.24.2.3 CNF

`using smtrat::cadcells::CNF = typedef std::vector<Disjunction>`

0.14.24.2.4 Constraint

`using smtrat::cadcells::Constraint = typedef carl::BasicConstraint<Polynomial>`

0.14.24.2.5 Disjunction

`using smtrat::cadcells::Disjunction = typedef std::vector<Atom>`

0.14.24.2.6 MultivariateRoot

`using smtrat::cadcells::MultivariateRoot = typedef carl::Multivariate↩ Root<Polynomial>`

0.14.24.2.7 Polynomial

`using smtrat::cadcells::Polynomial = typedef carl::ContextPolynomial<Rational>`

0.14.24.2.8 RAN

`using smtrat::cadcells::RAN = typedef Polynomial::RootType`

0.14.24.2.9 VariableComparison

`using smtrat::cadcells::VariableComparison = typedef carl::↩ VariableComparison<Polynomial>`

0.14.24.2.10 VariableOrdering

`using smtrat::cadcells::VariableOrdering = typedef std::vector<carl↩ ::Variable>`

0.14.24.3 Variable Documentation

0.14.24.3.1 empty_assignment

`const Assignment smtrat::cadcells::empty_assignment [static]`

0.14.25 smtrat::cadcells::algorithms Namespace Reference

Various algorithms as well as helper functions for developing new algorithms.

Functions

- template<cadcells::operators::op op, representation::CellHeuristic cell_heuristic>
  std::optional< std::pair< carl::Variable, datastructures::SymbolicInterval > > get_interval (datastructures::SampledDerivationR typename operators::PropertiesSet< op >::type > &cell_deriv)

  Single cell construction algorithm.
- template<cadcells::operators::op op, representation::CoveringHeuristic covering_heuristic>
  std::optional< datastructures::SampledDerivationRef< typename operators::PropertiesSet< op >::type > > get_level_covering (datastructures::Projections &proj, const std::vector< Atom > &constraints, const Assignment &sample)

  Computes an unsat covering w.r.t.
- template<cadcells::operators::op op>
  std::vector< datastructures::SampledDerivationRef< typename operators::PropertiesSet< op >::type > > get_unsat_intervals (const Constraint &c, datastructures::Projections &proj, const Assignment &sample)

- template<cadcells::operators::op op>
  std::vector< datastructures::SampledDerivationRef< typename operators::PropertiesSet< op >::type > > get_unsat_intervals (const VariableComparison &c, datastructures::Projections &proj, const Assignment &sample)
- template<cadcells::operators::op op>
  std::vector< datastructures::SampledDerivationRef< typename operators::PropertiesSet< op >::type > > get_unsat_intervals (const Atom &c, datastructures::Projections &proj, const Assignment &sample)

  Returns the unsat intervals of the given atom w.r.t.

0.14.25.1 Detailed Description

Various algorithms as well as helper functions for developing new algorithms.

0.14.25.2 Function Documentation

0.14.25.2.1 get_interval()

```
template<cadcells::operators::op op, representation::CellHeuristic cell_heuristic>
std::optional<std::pair<carl::Variable, datastructures::SymbolicInterval> > smtrat::cadcells↩
::algorithms::get_interval (
            datastructures::SampledDerivationRef< typename operators::PropertiesSet< op >↩
::type > & cell_deriv )
```

Single cell construction algorithm.

Template Parameters

op	The operator.
cell_heuristic	The cell heuristic.

Parameters

cell_deriv	A derivation object to construct the cell from.

Returns

A vector of pairs of variables and their symbolic intervals on success or std::nullopt otherwise.

0.14.25.2.2 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 std::vector< Atom > & constraints,
            const Assignment & sample )
```

Computes an unsat covering w.r.t.
a set of constraints.
If a constraint is univariate under a sample, for the 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.

0.14.25.2.3 get_unsat_intervals() [1/3]

```
template<cadcells::operators::op op>
std::vector<datastructures::SampledDerivationRef<typename operators::PropertiesSet<op>↩
::type> > smtrat::cadcells::algorithms::get_unsat_intervals (
            const Atom & c,
            datastructures::Projections & proj,
            const Assignment & sample )
```

Returns the unsat intervals of the given atom w.r.t.
an underlying sample.

Parameters

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.

0.14.25.2.4 get_unsat_intervals() [2/3]

```
template<cadcells::operators::op op>
std::vector<datastructures::SampledDerivationRef<typename operators::PropertiesSet<op>↩
::type> > smtrat::cadcells::algorithms::get_unsat_intervals (
            const Constraint & c,
            datastructures::Projections & proj,
            const Assignment & sample )
```

0.14.25.2.5 get_unsat_intervals() [3/3]

```
template<cadcells::operators::op op>
std::vector<datastructures::SampledDerivationRef<typename operators::PropertiesSet<op>↩
::type> > smtrat::cadcells::algorithms::get_unsat_intervals (
            const VariableComparison & c,
            datastructures::Projections & proj,
            const Assignment & sample )
```

0.14.26 smtrat::cadcells::datastructures Namespace Reference

Main datastructures.

Namespaces

- detail

Data Structures

- struct TaggedIndexedRoot
- class DelineationInterval

  An interval of a delineation.
- class Delineation

  Represents the delineation of a set of polynomials (at a sample), that is.

- class BaseDerivation

  A BaseDerivation has a level and a set of properties of this level, and an underlying derivation representing the lower levels.
- class DelineatedDerivation

  A DelineatedDerivation is a BaseDerivation with a Delineation and an underlying SampledDerivation.
- class SampledDerivation

  A SampledDerivation is a DelineatedDerivation with a sample and an DelineationInterval w.r.t.
- class DerivationRef

  A reference to a derivation, which is either.
- struct PolyRef

  Refers to a polynomial.
- class PolyPool

  A pool for polynomials.
- class Projections

  Encapsulates all computations on polynomials.
- struct property_hash
- struct PropertiesTContent
- struct PropertiesTContent< T, true >
- struct PropertiesTContent< T, false >
- struct PropertiesT

  Set of properties.
- struct PropertiesT< T, Ts... >
- struct CellRepresentation

  Represents a cell.
- struct CoveringRepresentation

  Represents a covering over a cell.
- struct IndexedRoot

  Represents the i-th root of a multivariate polynomial at its main variable (given an appropriate sample).
- struct CompoundMinMax

  Represents the minimum function of the contained indexed root functions.
- struct CompoundMaxMin

  Represents the maximum function of the contained indexed root functions.
- class RootFunction
- class Bound

  Bound type of a SymbolicInterval.
- class SymbolicInterval

  A symbolic interal whose bounds are defined by indexed root expressions.
- class CoveringDescription

  Describes a covering of the real line by SymbolicIntervals (given an appropriate sample).
- struct IndexedRootRelation

  A relation between two roots.
- class IndexedRootOrdering

  Describes an ordering of IndexedRoots.

Typedefs

- using RootMap = std::map< RAN, std::vector< TaggedIndexedRoot > >
- using RootMapPlain = std::map< RAN, std::vector< IndexedRoot > >
- template<typename Properties >
  using BaseDerivationRef = std::shared_ptr< BaseDerivation< Properties > >
- template<typename Properties >
  using DelineatedDerivationRef = std::shared_ptr< DelineatedDerivation< Properties > >
- template<typename Properties >
  using SampledDerivationRef = std::shared_ptr< SampledDerivation< Properties > >
- template<typename T >
  using PropertiesTSet = boost::container::flat_set< T >

Functions

- std::ostream & operator<< (std::ostream &os, const TaggedIndexedRoot &data)
- bool operator== (const TaggedIndexedRoot &lhs, const TaggedIndexedRoot &rhs)
- bool operator< (const TaggedIndexedRoot &lhs, const TaggedIndexedRoot &rhs)
- std::ostream & operator<< (std::ostream &os, const DelineationInterval &data)
- std::ostream & operator<< (std::ostream &os, const Delineation &data)
- bool lower_lt_lower (const DelineationInterval &del1, const DelineationInterval &del2)

  Compares the lower bounds of two DelineationIntervals.
- bool lower_eq_lower (const DelineationInterval &del1, const DelineationInterval &del2)

  Compares the lower bounds of two DelineationIntervals.
- bool upper_lt_upper (const DelineationInterval &del1, const DelineationInterval &del2)

  Compares the upper bounds of two DelineationIntervals.
- bool upper_lt_lower (const DelineationInterval &del1, const DelineationInterval &del2)

  Compares an upper bound with a lower bound of DelineationIntervals.
- template<typename P >
  bool operator== (const DerivationRef< P > &lhs, const DerivationRef< P > &rhs)
- template<typename P >
  bool operator< (const DerivationRef< P > &lhs, const DerivationRef< P > &rhs)
- template<typename Properties >
  DerivationRef< Properties > make_derivation (Projections &proj, const Assignment &assignment, size_↩t level)

  Initializes a derivation according the the given assignment and level.
- template<typename Properties >
  SampledDerivationRef< Properties > make_sampled_derivation (DelineatedDerivationRef< Properties > delineated, const RAN &main_sample)

  Initializes a sampled derivation w.r.t.
- template<typename Properties >
  void merge_underlying (std::vector< SampledDerivationRef< Properties >> &derivations)

  Merges the underlying derivations of a set of sampled derivations.
- bool operator< (const PolyRef &lhs, const PolyRef &rhs)
- bool operator== (const PolyRef &lhs, const PolyRef &rhs)
- bool operator!= (const PolyRef &lhs, const PolyRef &rhs)
- std::ostream & operator<< (std::ostream &os, const PolyRef &data)
- template<class T , class... Ts>
  void prop_insert (PropertiesT< T, Ts... > &sets, const T &element)
- template<class S , class T , class... Ts, typename std::enable_if<!std::is_same< S, T >::value >::type >
  void prop_insert (PropertiesT< T, Ts... > &sets, const S &element)
- template<class T , class... Ts>
  bool prop_has (const PropertiesT< T, Ts... > &sets, const T &element)
- template<class S , class T , class... Ts, typename std::enable_if<!std::is_same< S, T >::value >::type >
  bool prop_has (const PropertiesT< T, Ts... > &sets, const S &element)
- template<class T , class... Ts>
  const auto & prop_get (const PropertiesT< T, Ts... > &sets)
- template<class S , class T , class... Ts, typename std::enable_if<!std::is_same< S, T >::value >::type >
  const auto & prop_get (const PropertiesT< T, Ts... > &sets)
- template<class T , class... Ts>
  void merge (PropertiesT< T, Ts... > &sets_a, const PropertiesT< T, Ts... > &sets_b)
- template<typename P >
  std::ostream & operator<< (std::ostream &os, const CellRepresentation< P > &data)
- template<typename P >
  std::ostream & operator<< (std::ostream &os, const CoveringRepresentation< P > &data)
- bool operator== (const IndexedRoot &lhs, const IndexedRoot &rhs)
- bool operator< (const IndexedRoot &lhs, const IndexedRoot &rhs)
- bool operator!= (const IndexedRoot &lhs, const IndexedRoot &rhs)

- std::ostream & operator<< (std::ostream &os, const IndexedRoot &data)
- bool operator== (const CompoundMinMax &lhs, const CompoundMinMax &rhs)
- bool operator< (const CompoundMinMax &lhs, const CompoundMinMax &rhs)
- bool operator!= (const CompoundMinMax &lhs, const CompoundMinMax &rhs)
- std::ostream & operator<< (std::ostream &os, const CompoundMinMax &data)
- bool operator== (const CompoundMaxMin &lhs, const CompoundMaxMin &rhs)
- bool operator< (const CompoundMaxMin &lhs, const CompoundMaxMin &rhs)
- bool operator!= (const CompoundMaxMin &lhs, const CompoundMaxMin &rhs)
- std::ostream & operator<< (std::ostream &os, const CompoundMaxMin &data)
- bool operator== (const RootFunction &lhs, const RootFunction &rhs)
- bool operator< (const RootFunction &lhs, const RootFunction &rhs)
- bool operator!= (const RootFunction &lhs, const RootFunction &rhs)
- std::ostream & operator<< (std::ostream &os, const RootFunction &data)
- std::ostream & operator<< (std::ostream &os, const SymbolicInterval &data)
- std::ostream & operator<< (std::ostream &os, const CoveringDescription &data)
- bool operator== (const IndexedRootRelation &lhs, const IndexedRootRelation &rhs)
- bool operator< (const IndexedRootRelation &lhs, const IndexedRootRelation &rhs)
- std::ostream & operator<< (std::ostream &os, const IndexedRootRelation &data)
- std::ostream & operator<< (std::ostream &os, const IndexedRootOrdering &data)

0.14.26.1 Detailed Description

Main datastructures.

0.14.26.1.1 Polynomials and projection results We assume a fixed variable ordering VariableOrdering. All polynomials are pooled in PolyPool w.r.t. this ordering, that is, they are identified by a PolyRef (a pair of the polynomial's level and an ID on this level). Projection results are cached in Projections. The latter holds a reference to a PolyPool. An instance Projections will be required for all kinds of operations and passed during various function calls. Thus, we initialize these data structures as follows:

```
VariableOrdering vars; // a variable ordering of all variables that will occur.
datastructures::PolyPool pool(vars);
datastructures::Projections proj(pool);
```

Note that PolyPool and Projections do not do reference counting; the cache needs to be cleared explicitly, see Projections::clear_cache and Projections::clear_assignment_cache.

0.14.26.1.2 Basic datastructures The basic datastructures for representing mathematical objects are datastructures::IndexedRoot, datastructures::SymbolicInterval, datastructures::CoveringDescription, and datastructures::IndexedRoot

0.14.26.1.3 Properties datastructures::PropertiesT is the datastructure for storing properties on a single level.
If you work on an algorithm using this framework, you don't need to know this datastructure. If you need to implement a new property, then you find the interfaces a property needs to implement there.

0.14.26.1.4 Delineations datastructures::Delineation is the datastructure for storing the delineation of the indexed roots that stem from properties.
If you work on an algorithm using this framework, you don't need to know this datastructure. If you need to implement a new property or a heuristic for describing a cell (i.e. compute a representation), then you might be interested in this.

0.14.26.1.5 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.

0.14.26.1.6 Representations The derivation objects do not store heuristic decisions, they just describe the current state. At some point, a representation of this state needs to be determined that is passed to the operator. This heuristic decision is stored in datastructures::CellRepresentation or datastructures::CoveringRepresentation.
The heuristics for computing representations are in representation.

0.14.26.1.7 Operators The operators work on derivations and representations.

0.14.26.2 Typedef Documentation

0.14.26.2.1 BaseDerivationRef

```
template<typename Properties >
using smtrat::cadcells::datastructures::BaseDerivationRef = typedef std::shared_ptr<BaseDerivation<Properties
>
```

0.14.26.2.2 DelineatedDerivationRef

```
template<typename Properties >
using smtrat::cadcells::datastructures::DelineatedDerivationRef = typedef std::shared_ptr<DelineatedDerivatio
>
```

0.14.26.2.3 PropertiesTSet

```
template<typename T >
using smtrat::cadcells::datastructures::PropertiesTSet = typedef boost::container::flat_set<T>
```

0.14.26.2.4 RootMap

```
using smtrat::cadcells::datastructures::RootMap = typedef std::map<RAN,
std::vector<TaggedIndexedRoot> >
```

0.14.26.2.5 RootMapPlain

```
using smtrat::cadcells::datastructures::RootMapPlain = typedef std↩
::map<RAN, std::vector<IndexedRoot> >
```

0.14.26.2.6 SampledDerivationRef

```
template<typename Properties >
using smtrat::cadcells::datastructures::SampledDerivationRef = typedef std::shared_ptr<SampledDerivation<Prop
>
```

0.14.26.3 Function Documentation

0.14.26.3.1 lower_eq_lower()

```
bool smtrat::cadcells::datastructures::lower_eq_lower (
    const DelineationInterval & del1,
    const DelineationInterval & del2 ) [inline]
```

Compares the lower bounds of two DelineationIntervals.
It respects whether the interval is a section or sector.

0.14.26.3.2 lower_lt_lower()

```
bool smtrat::cadcells::datastructures::lower_lt_lower (
    const DelineationInterval & del1,
    const DelineationInterval & del2 ) [inline]
```

Compares the lower bounds of two DelineationIntervals.
It respects whether the interval is a section or sector.

0.14.26.3.3 make_derivation()

```
template<typename Properties >
DerivationRef<Properties> smtrat::cadcells::datastructures::make_derivation (
    Projections & proj,
    const Assignment & assignment,
    size_t level )
```

Initializes a derivation according the the given assignment and level.

0.14.26.3.4 make_sampled_derivation()

```
template<typename Properties >
SampledDerivationRef<Properties> smtrat::cadcells::datastructures::make_sampled_derivation (
    DelineatedDerivationRef< Properties > delineated,
    const RAN & main_sample )
```

Initializes a sampled derivation w.r.t.
the delineated derivation and sample.

0.14.26.3.5 merge()

```
template<class T , class... Ts>
void smtrat::cadcells::datastructures::merge (
    PropertiesT< T, Ts... > & sets_a,
    const PropertiesT< T, Ts... > & sets_b )
```

0.14.26.3.6 merge_underlying()

```
template<typename Properties >
void smtrat::cadcells::datastructures::merge_underlying (
    std::vector< SampledDerivationRef< Properties >> & derivations )
```

Merges the underlying derivations of a set of sampled derivations.
After the operation, all sampled derivations point to the same underlying derivation.

0.14.26.3.7 operator"!=() [1/5]

```
bool smtrat::cadcells::datastructures::operator!= (
    const CompoundMaxMin & lhs,
    const CompoundMaxMin & rhs ) [inline]
```

0.14.26.3.8 operator"!=() [2/5]

```
bool smtrat::cadcells::datastructures::operator!= (
    const CompoundMinMax & lhs,
    const CompoundMinMax & rhs ) [inline]
```

0.14.26.3.9 operator"!=() [3/5]

```
bool smtrat::cadcells::datastructures::operator!= (
    const IndexedRoot & lhs,
    const IndexedRoot & rhs ) [inline]
```

0.14.26.3.10 operator"!=() [4/5]

```
bool smtrat::cadcells::datastructures::operator!= (
    const PolyRef & lhs,
    const PolyRef & rhs ) [inline]
```

0.14.26.3.11 operator"!=() [5/5]

```
bool smtrat::cadcells::datastructures::operator!= (
    const RootFunction & lhs,
    const RootFunction & rhs ) [inline]
```

0.14.26.3.12 operator<() [1/8]

```
bool smtrat::cadcells::datastructures::operator< (
    const CompoundMaxMin & lhs,
    const CompoundMaxMin & rhs ) [inline]
```

0.14.26.3.13 operator<() [2/8]

```
bool smtrat::cadcells::datastructures::operator< (
    const CompoundMinMax & lhs,
    const CompoundMinMax & rhs ) [inline]
```

0.14.26.3.14 operator<() [3/8]

```
template<typename P >
bool smtrat::cadcells::datastructures::operator< (
    const DerivationRef< P > & lhs,
    const DerivationRef< P > & rhs )
```

0.14.26.3.15 operator<() [4/8]

```
bool smtrat::cadcells::datastructures::operator< (
    const IndexedRoot & lhs,
    const IndexedRoot & rhs ) [inline]
```

0.14.26.3.16 operator<() [5/8]

```
bool smtrat::cadcells::datastructures::operator< (
    const IndexedRootRelation & lhs,
    const IndexedRootRelation & rhs ) [inline]
```

0.14.26.3.17 operator<() [6/8]

```
bool smtrat::cadcells::datastructures::operator< (
    const PolyRef & lhs,
    const PolyRef & rhs ) [inline]
```

0.14.26.3.18 operator<() [7/8]

```
bool smtrat::cadcells::datastructures::operator< (
    const RootFunction & lhs,
    const RootFunction & rhs ) [inline]
```

0.14.26.3.19 operator<() [8/8]

```
bool smtrat::cadcells::datastructures::operator< (
    const TaggedIndexedRoot & lhs,
    const TaggedIndexedRoot & rhs ) [inline]
```

0.14.26.3.20 operator<<() [1/14]

```
template<typename P >
std::ostream& smtrat::cadcells::datastructures::operator<< (
    std::ostream & os,
    const CellRepresentation< P > & data ) [inline]
```

0.14.26.3.21 operator<<() [2/14] std::ostream& smtrat::cadcells::datastructures::operator<< (
std::ostream & *os*,
const CompoundMaxMin & *data*) [inline]

0.14.26.3.22 operator<<() [3/14] std::ostream& smtrat::cadcells::datastructures::operator<< (
std::ostream & *os*,
const CompoundMinMax & *data*) [inline]

0.14.26.3.23 operator<<() [4/14] std::ostream& smtrat::cadcells::datastructures::operator<< (
std::ostream & *os*,
const CoveringDescription & *data*) [inline]

0.14.26.3.24 operator<<() [5/14] template<typename P >
std::ostream& smtrat::cadcells::datastructures::operator<< (
std::ostream & *os*,
const CoveringRepresentation< P > & *data*)

0.14.26.3.25 operator<<() [6/14] std::ostream& smtrat::cadcells::datastructures::operator<< (
std::ostream & *os*,
const Delineation & *data*) [inline]

0.14.26.3.26 operator<<() [7/14] std::ostream& smtrat::cadcells::datastructures::operator<< (
std::ostream & *os*,
const DelineationInterval & *data*) [inline]

0.14.26.3.27 operator<<() [8/14] std::ostream& smtrat::cadcells::datastructures::operator<< (
std::ostream & *os*,
const IndexedRoot & *data*) [inline]

0.14.26.3.28 operator<<() [9/14] std::ostream& smtrat::cadcells::datastructures::operator<< (
std::ostream & *os*,
const IndexedRootOrdering & *data*) [inline]

0.14.26.3.29 operator<<() [10/14] std::ostream& smtrat::cadcells::datastructures::operator<< (
std::ostream & *os*,
const IndexedRootRelation & *data*) [inline]

0.14.26.3.30 operator<<() [11/14] std::ostream& smtrat::cadcells::datastructures::operator<< (
std::ostream & *os*,
const PolyRef & *data*) [inline]

0.14.26.3.31 operator<<() [12/14] std::ostream& smtrat::cadcells::datastructures::operator<< (
std::ostream & *os*,
const RootFunction & *data*) [inline]

0.14.26.3.32 operator<<() [13/14]

```
std::ostream& smtrat::cadcells::datastructures::operator<< (
    std::ostream & os,
    const SymbolicInterval & data ) [inline]
```

0.14.26.3.33 operator<<() [14/14]

```
std::ostream& smtrat::cadcells::datastructures::operator<< (
    std::ostream & os,
    const TaggedIndexedRoot & data ) [inline]
```

0.14.26.3.34 operator==() [1/8]

```
bool smtrat::cadcells::datastructures::operator== (
    const CompoundMaxMin & lhs,
    const CompoundMaxMin & rhs ) [inline]
```

0.14.26.3.35 operator==() [2/8]

```
bool smtrat::cadcells::datastructures::operator== (
    const CompoundMinMax & lhs,
    const CompoundMinMax & rhs ) [inline]
```

0.14.26.3.36 operator==() [3/8]

```
template<typename P >
bool smtrat::cadcells::datastructures::operator== (
    const DerivationRef< P > & lhs,
    const DerivationRef< P > & rhs )
```

0.14.26.3.37 operator==() [4/8]

```
bool smtrat::cadcells::datastructures::operator== (
    const IndexedRoot & lhs,
    const IndexedRoot & rhs ) [inline]
```

0.14.26.3.38 operator==() [5/8]

```
bool smtrat::cadcells::datastructures::operator== (
    const IndexedRootRelation & lhs,
    const IndexedRootRelation & rhs ) [inline]
```

0.14.26.3.39 operator==() [6/8]

```
bool smtrat::cadcells::datastructures::operator== (
    const PolyRef & lhs,
    const PolyRef & rhs ) [inline]
```

0.14.26.3.40 operator==() [7/8]

```
bool smtrat::cadcells::datastructures::operator== (
    const RootFunction & lhs,
    const RootFunction & rhs ) [inline]
```

0.14.26.3.41 operator==() [8/8]

```
bool smtrat::cadcells::datastructures::operator== (
    const TaggedIndexedRoot & lhs,
    const TaggedIndexedRoot & rhs ) [inline]
```

0.14.26.3.42 prop_get() [1/2]

```
template<class T , class... Ts>
const auto& smtrat::cadcells::datastructures::prop_get (
    const PropertiesT< T, Ts... > & sets )
```

0.14.26.3.43 prop_get() [2/2]

```
template<class S , class T , class... Ts, typename std::enable↩
_if<!std::is_same< S, T >::value >::type >
const auto& smtrat::cadcells::datastructures::prop_get (
            const PropertiesT< T, Ts... > & sets )
```

0.14.26.3.44 prop_has() [1/2]

```
template<class S , class T , class... Ts, typename std::enable↩
_if<!std::is_same< S, T >::value >::type >
bool smtrat::cadcells::datastructures::prop_has (
            const PropertiesT< T, Ts... > & sets,
            const S & element )
```

0.14.26.3.45 prop_has() [2/2]

```
template<class T , class... Ts>
bool smtrat::cadcells::datastructures::prop_has (
            const PropertiesT< T, Ts... > & sets,
            const T & element )
```

0.14.26.3.46 prop_insert() [1/2]

```
template<class S , class T , class... Ts, typename std::enable↩
_if<!std::is_same< S, T >::value >::type >
void smtrat::cadcells::datastructures::prop_insert (
            PropertiesT< T, Ts... > & sets,
            const S & element )
```

0.14.26.3.47 prop_insert() [2/2]

```
template<class T , class... Ts>
void smtrat::cadcells::datastructures::prop_insert (
            PropertiesT< T, Ts... > & sets,
            const T & element )
```

0.14.26.3.48 upper_lt_lower()

```
bool smtrat::cadcells::datastructures::upper_lt_lower (
            const DelineationInterval & del1,
            const DelineationInterval & del2 ) [inline]
```

Compares an upper bound with a lower bound of DelineationIntervals.
It respects whether the interval is a section or sector.

0.14.26.3.49 upper_lt_upper()

```
bool smtrat::cadcells::datastructures::upper_lt_upper (
            const DelineationInterval & del1,
            const DelineationInterval & del2 ) [inline]
```

Compares the upper bounds of two DelineationIntervals.
It respects whether the interval is a section or sector.

0.14.27 smtrat::cadcells::datastructures::detail Namespace Reference

Data Structures

- struct PolyProperties
- struct AssignmentProperties

0.14.28 smtrat::cadcells::helper Namespace Reference

Functions

- MultivariateRoot as_multivariate_root (const datastructures::PolyPool &pool, carl::Variable main_var, datastructures::IndexedRoot r)

Converts an datastructures::IndexedRoot to a MultivariateRoot.

- CNF to_formula (const datastructures::PolyPool &pool, carl::Variable main_var, const datastructures::SymbolicInterval &c)

  Converts a datastructures::SymbolicInterval to a CNF.

0.14.28.1 Function Documentation

0.14.28.1.1 as_multivariate_root()

```
MultivariateRoot smtrat::cadcells::helper::as_multivariate_root (
    const datastructures::PolyPool & pool,
    carl::Variable main_var,
    datastructures::IndexedRoot r ) [inline]
```

Converts an datastructures::IndexedRoot to a MultivariateRoot.

0.14.28.1.2 to_formula()

```
CNF smtrat::cadcells::helper::to_formula (
    const datastructures::PolyPool & pool,
    carl::Variable main_var,
    const datastructures::SymbolicInterval & c ) [inline]
```

Converts a datastructures::SymbolicInterval to a CNF.

0.14.29 smtrat::cadcells::operators Namespace Reference

Projection operators.

Namespaces

- mccallum_filtered_impl
- properties

  Contains all properties that are stored in a derivation.
- rules

  Implementation of derivation rules.

Data Structures

- struct PropertiesSet
- struct PropertiesSet< op::mccallum >

Enumerations

- enum op {
  mccallum , mccallum_filtered , mccallum_filtered_all , mccallum_filtered_bounds ,
  mccallum_filtered_samples , mccallum_filtered_all_selective , mccallum_filtered_all_compound }

Functions

- std::ostream & operator<< (std::ostream &os, op op)
- template<op Op>
  bool project_basic_properties (datastructures::DelineatedDerivation< typename PropertiesSet< Op >::type > &deriv)

  Project basic properties, i.e.
- template<op Op>
  void delineate_properties (datastructures::DelineatedDerivation< typename PropertiesSet< Op >::type > &deriv)

  Delineate properties.

- template<op Op>
  void delineate_properties (datastructures::SampledDerivation< typename PropertiesSet< Op >::type > &deriv)

  Delineate properties.
- template<op Op>
  bool project_delineated_cell_properties (datastructures::CellRepresentation< typename PropertiesSet< Op >::type > &deriv, bool cell_represents=true)

  Project cell properties that depend on a delineation.
- template<op Op>
  bool project_cell_properties (datastructures::SampledDerivation< typename PropertiesSet< Op >::type > &deriv)

  Project cell properties.
- template<op Op>
  bool project_covering_properties (datastructures::CoveringRepresentation< typename PropertiesSet< Op >::type > &repr)

  Project covering properties.
- template<> bool project_basic_properties< op::mccallum > (datastructures::DelineatedDerivation< PropertiesSet< op::mccallum >::type > &deriv)
- template<> void delineate_properties< op::mccallum > (datastructures::DelineatedDerivation< PropertiesSet< op::mccallum >::type > &deriv)
- template<> void delineate_properties< op::mccallum > (datastructures::SampledDerivation< PropertiesSet< op::mccallum >::type > &deriv)
- template<> bool project_delineated_cell_properties< op::mccallum > (datastructures::CellRepresentation< PropertiesSet< op::mccallum >::type > &repr, bool cell_represents)
- template<> bool project_cell_properties< op::mccallum > (datastructures::SampledDerivation< PropertiesSet< op::mccallum >::type > &deriv)
- template<> bool project_covering_properties< op::mccallum > (datastructures::CoveringRepresentation< PropertiesSet< op::mccallum >::type > &repr)
- template<> void delineate_properties< op::mccallum_filtered > (datastructures::SampledDerivation< PropertiesSet< op::mccallum_filtered >::type > &deriv)
- template<> void delineate_properties< op::mccallum_filtered_all > (datastructures::SampledDerivation< PropertiesSet< op::mccallum_filtered_all >::type > &deriv)
- template<> void delineate_properties< op::mccallum_filtered_bounds > (datastructures::SampledDerivation< PropertiesSet< op::mccallum_filtered_bounds >::type > &deriv)
- template<> void delineate_properties< op::mccallum_filtered_samples > (datastructures::SampledDerivation< PropertiesSet< op::mccallum_filtered_samples >::type > &deriv)
- template<> void delineate_properties< op::mccallum_filtered_all_selective > (datastructures::SampledDerivation< PropertiesSet< op::mccallum_filtered_all_selective >::type > &deriv)
- template<> void delineate_properties< op::mccallum_filtered_all_compound > (datastructures::SampledDerivation< PropertiesSet< op::mccallum_filtered_all_compound >::type > &deriv)

Variables

- static const char * OpStrings [ ] = { "mccallum", "mccallum_filtered", "mccallum_filtered_all", "mccallum_filtered_bounds", "mccallum_filtered_samples", "mccallum_filtered_all_selective" }

0.14.29.1 Detailed Description

Projection operators.
Currently, only the McCallum based operator is implemented.

0.14.29.1.1 Relevant projection functions

We refer to algorithms for usage examples.

0.14.29.1.1.1 Cells

- project_cell_properties
- project_basic_properties
- delineate_properties
- project_delineated_cell_properties

0.14.29.1.1.2 Coverings

- project_cell_properties (on each cell individually)
- project_basic_properties (on each cell individually)
- delineate_properties (on each cell individually)
- project_covering_properties

0.14.29.1.1.3 Delineation

- project_basic_properties
- delineate_properties
- project_delineation_properties

0.14.29.2 Enumeration Type Documentation

0.14.29.2.1 op `enum smtrat::cadcells::operators::op`

Enumerator

mccallum	
mccallum_filtered	
mccallum_filtered_all	
mccallum_filtered_bounds	
mccallum_filtered_samples	
mccallum_filtered_all_selective	
mccallum_filtered_all_compound	

0.14.29.3 Function Documentation

0.14.29.3.1 delineate_properties() [1/2] `template<op Op>`

```
void smtrat::cadcells::operators::delineate_properties (
            datastructures::DelineatedDerivation< typename PropertiesSet< Op >::type > &
deriv ) [inline]
```

Delineate properties.

0.14.29.3.2 delineate_properties() [2/2] `template<op Op>`

```
void smtrat::cadcells::operators::delineate_properties (
            datastructures::SampledDerivation< typename PropertiesSet< Op >::type > & deriv
) [inline]
```

Delineate properties.

0.14.29.3.3 delineate_properties< op::mccallum >() [1/2] template<>

```
void smtrat::cadcells::operators::delineate_properties< op::mccallum > (
            datastructures::DelineatedDerivation< PropertiesSet< op::mccallum >::type > &
deriv )  [inline]
```

0.14.29.3.4 delineate_properties< op::mccallum >() [2/2] template<>

```
void smtrat::cadcells::operators::delineate_properties< op::mccallum > (
            datastructures::SampledDerivation< PropertiesSet< op::mccallum >::type > & deriv
)  [inline]
```

0.14.29.3.5 delineate_properties< op::mccallum_filtered >() template<>

```
void smtrat::cadcells::operators::delineate_properties< op::mccallum_filtered > (
            datastructures::SampledDerivation< PropertiesSet< op::mccallum_filtered >::type
> & deriv )  [inline]
```

0.14.29.3.6 delineate_properties< op::mccallum_filtered_all >() template<>

```
void smtrat::cadcells::operators::delineate_properties< op::mccallum_filtered_all > (
            datastructures::SampledDerivation< PropertiesSet< op::mccallum_filtered_all >↩
::type > & deriv )  [inline]
```

0.14.29.3.7 delineate_properties< op::mccallum_filtered_all_compound >() template<>

```
void smtrat::cadcells::operators::delineate_properties< op::mccallum_filtered_all_compound > (
            datastructures::SampledDerivation< PropertiesSet< op::mccallum_filtered_all_↩
compound >::type > & deriv )  [inline]
```

0.14.29.3.8 delineate_properties< op::mccallum_filtered_all_selective >() template<>

```
void smtrat::cadcells::operators::delineate_properties< op::mccallum_filtered_all_selective >
(
            datastructures::SampledDerivation< PropertiesSet< op::mccallum_filtered_all_↩
selective >::type > & deriv )  [inline]
```

0.14.29.3.9 delineate_properties< op::mccallum_filtered_bounds >() template<>

```
void smtrat::cadcells::operators::delineate_properties< op::mccallum_filtered_bounds > (
            datastructures::SampledDerivation< PropertiesSet< op::mccallum_filtered_bounds
>::type > & deriv )  [inline]
```

0.14.29.3.10 delineate_properties< op::mccallum_filtered_samples >() template<>

```
void smtrat::cadcells::operators::delineate_properties< op::mccallum_filtered_samples > (
            datastructures::SampledDerivation< PropertiesSet< op::mccallum_filtered_samples
>::type > & deriv )  [inline]
```

0.14.29.3.11 operator<<() std::ostream& smtrat::cadcells::operators::operator<< (

```
            std::ostream & os,
            op op )  [inline]
```

0.14.29.3.12 project_basic_properties() template<op Op>

```
bool smtrat::cadcells::operators::project_basic_properties (
            datastructures::DelineatedDerivation< typename PropertiesSet< Op >::type > &
deriv ) [inline]
```

Project basic properties, i.e.
independent from any sample or delineation.

0.14.29.3.13 project_basic_properties< op::mccallum >() template<>

```
bool smtrat::cadcells::operators::project_basic_properties< op::mccallum > (
            datastructures::DelineatedDerivation< PropertiesSet< op::mccallum >::type > &
deriv ) [inline]
```

0.14.29.3.14 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).

0.14.29.3.15 project_cell_properties< op::mccallum >() template<>

```
bool smtrat::cadcells::operators::project_cell_properties< op::mccallum > (
            datastructures::SampledDerivation< PropertiesSet< op::mccallum >::type > & deriv
) [inline]
```

0.14.29.3.16 project_covering_properties() template<op Op>

```
bool smtrat::cadcells::operators::project_covering_properties (
            datastructures::CoveringRepresentation< typename PropertiesSet< Op >::type > &
repr ) [inline]
```

Project covering properties.

0.14.29.3.17 project_covering_properties< op::mccallum >() template<>

```
bool smtrat::cadcells::operators::project_covering_properties< op::mccallum > (
            datastructures::CoveringRepresentation< PropertiesSet< op::mccallum >::type > &
repr ) [inline]
```

0.14.29.3.18 project_delineated_cell_properties() template<op Op>

```
bool smtrat::cadcells::operators::project_delineated_cell_properties (
            datastructures::CellRepresentation< typename PropertiesSet< Op >::type > & deriv,
            bool cell_represents = true ) [inline]
```

Project cell properties that depend on a delineation.

0.14.29.3.19 project_delineated_cell_properties< op::mccallum >() template<>

```
bool smtrat::cadcells::operators::project_delineated_cell_properties< op::mccallum > (
            datastructures::CellRepresentation< PropertiesSet< op::mccallum >::type > &
repr,
            bool cell_represents ) [inline]
```

0.14.29.4 Variable Documentation

0.14.29.4.1 OpStrings

`const char* smtrat::cadcells::operators::OpStrings[] = { "mccallum", "mccallum_filtered", "mccallum_filtered_all", "mccallum_filtered_bounds", "mccallum_filtered_samples", "mccallum_filtered_all_selective" } [static]`

0.14.30 smtrat::cadcells::operators::mccallum_filtered_impl Namespace Reference

Data Structures

- struct PropertiesSet

Functions

- bool project_basic_properties (datastructures::DelineatedDerivation< PropertiesSet::type > &deriv)
- void delineate_properties (datastructures::DelineatedDerivation< PropertiesSet::type > &deriv)
- bool project_delineated_cell_properties (datastructures::CellRepresentation< PropertiesSet::type > &repr, bool cell_represents)
- bool project_cell_properties (datastructures::SampledDerivation< PropertiesSet::type > &deriv)
- bool project_covering_properties (datastructures::CoveringRepresentation< PropertiesSet::type > &repr)

0.14.30.1 Function Documentation

0.14.30.1.1 delineate_properties()

```
void smtrat::cadcells::operators::mccallum_filtered_impl↩
::delineate_properties (
            datastructures::DelineatedDerivation< PropertiesSet::type > & deriv ) [inline]
```

0.14.30.1.2 project_basic_properties()

```
bool smtrat::cadcells::operators::mccallum_filtered_impl↩
::project_basic_properties (
            datastructures::DelineatedDerivation< PropertiesSet::type > & deriv ) [inline]
```

0.14.30.1.3 project_cell_properties()

```
bool smtrat::cadcells::operators::mccallum_filtered_impl↩
::project_cell_properties (
            datastructures::SampledDerivation< PropertiesSet::type > & deriv ) [inline]
```

0.14.30.1.4 project_covering_properties()

```
bool smtrat::cadcells::operators::mccallum_filtered_↩
impl::project_covering_properties (
            datastructures::CoveringRepresentation< PropertiesSet::type > & repr ) [inline]
```

0.14.30.1.5 project_delineated_cell_properties()

```
bool smtrat::cadcells::operators::mccallum_↩
filtered_impl::project_delineated_cell_properties (
            datastructures::CellRepresentation< PropertiesSet::type > & repr,
            bool cell_represents ) [inline]
```

0.14.31 smtrat::cadcells::operators::properties Namespace Reference

Contains all properties that are stored in a derivation.

Data Structures

- struct poly_sgn_inv
- struct poly_irreducible_sgn_inv
- struct poly_semi_sgn_inv

- struct poly_irreducible_semi_sgn_inv
- struct poly_ord_inv
- struct root_well_def
- struct poly_pdel
- struct cell_connected
- struct poly_additional_root_outside
- struct poly_ord_inv_base
- struct root_ordering_holds

Functions

- bool operator== (const poly_sgn_inv &lhs, const poly_sgn_inv &rhs)
- bool operator< (const poly_sgn_inv &lhs, const poly_sgn_inv &rhs)
- std::ostream & operator<< (std::ostream &os, const poly_sgn_inv &data)
- bool operator== (const poly_irreducible_sgn_inv &lhs, const poly_irreducible_sgn_inv &rhs)
- bool operator< (const poly_irreducible_sgn_inv &lhs, const poly_irreducible_sgn_inv &rhs)
- std::ostream & operator<< (std::ostream &os, const poly_irreducible_sgn_inv &data)
- bool operator== (const poly_semi_sgn_inv &lhs, const poly_semi_sgn_inv &rhs)
- bool operator< (const poly_semi_sgn_inv &lhs, const poly_semi_sgn_inv &rhs)
- std::ostream & operator<< (std::ostream &os, const poly_semi_sgn_inv &data)
- bool operator== (const poly_irreducible_semi_sgn_inv &lhs, const poly_irreducible_semi_sgn_inv &rhs)
- bool operator< (const poly_irreducible_semi_sgn_inv &lhs, const poly_irreducible_semi_sgn_inv &rhs)
- std::ostream & operator<< (std::ostream &os, const poly_irreducible_semi_sgn_inv &data)
- bool operator== (const poly_ord_inv &lhs, const poly_ord_inv &rhs)
- bool operator< (const poly_ord_inv &lhs, const poly_ord_inv &rhs)
- std::ostream & operator<< (std::ostream &os, const poly_ord_inv &data)
- bool operator== (const root_well_def &lhs, const root_well_def &rhs)
- bool operator< (const root_well_def &lhs, const root_well_def &rhs)
- std::ostream & operator<< (std::ostream &os, const root_well_def &data)
- bool operator== (const poly_pdel &lhs, const poly_pdel &rhs)
- bool operator< (const poly_pdel &lhs, const poly_pdel &rhs)
- std::ostream & operator<< (std::ostream &os, const poly_pdel &data)
- bool operator== (const cell_connected &lhs, const cell_connected &rhs)
- bool operator< (const cell_connected &lhs, const cell_connected &rhs)
- std::ostream & operator<< (std::ostream &os, const cell_connected &data)
- bool operator== (const poly_additional_root_outside &lhs, const poly_additional_root_outside &rhs)
- bool operator< (const poly_additional_root_outside &lhs, const poly_additional_root_outside &rhs)
- std::ostream & operator<< (std::ostream &os, const poly_additional_root_outside &data)
- bool operator== (const poly_ord_inv_base &lhs, const poly_ord_inv_base &rhs)
- bool operator< (const poly_ord_inv_base &lhs, const poly_ord_inv_base &rhs)
- std::ostream & operator<< (std::ostream &os, const poly_ord_inv_base &data)
- bool operator== (const root_ordering_holds &lhs, const root_ordering_holds &rhs)
- bool operator< (const root_ordering_holds &lhs, const root_ordering_holds &rhs)
- std::ostream & operator<< (std::ostream &os, const root_ordering_holds &data)

0.14.31.1 Detailed Description

Contains all properties that are stored in a derivation.
Note that not all properties have a representation here as not all of them are stored but resolved directly in the derivation rules.

0.14.31.2 Function Documentation

0.14.31.2.1 operator<() [1/11]

```
bool smtrat::cadcells::operators::properties::operator< (
    const cell_connected & lhs,
    const cell_connected & rhs ) [inline]
```

0.14.31.2.2 operator<() [2/11]

```
bool smtrat::cadcells::operators::properties::operator< (
    const poly_additional_root_outside & lhs,
    const poly_additional_root_outside & rhs ) [inline]
```

0.14.31.2.3 operator<() [3/11]

```
bool smtrat::cadcells::operators::properties::operator< (
    const poly_irreducible_semi_sgn_inv & lhs,
    const poly_irreducible_semi_sgn_inv & rhs ) [inline]
```

0.14.31.2.4 operator<() [4/11]

```
bool smtrat::cadcells::operators::properties::operator< (
    const poly_irreducible_sgn_inv & lhs,
    const poly_irreducible_sgn_inv & rhs ) [inline]
```

0.14.31.2.5 operator<() [5/11]

```
bool smtrat::cadcells::operators::properties::operator< (
    const poly_ord_inv & lhs,
    const poly_ord_inv & rhs ) [inline]
```

0.14.31.2.6 operator<() [6/11]

```
bool smtrat::cadcells::operators::properties::operator< (
    const poly_ord_inv_base & lhs,
    const poly_ord_inv_base & rhs ) [inline]
```

0.14.31.2.7 operator<() [7/11]

```
bool smtrat::cadcells::operators::properties::operator< (
    const poly_pdel & lhs,
    const poly_pdel & rhs ) [inline]
```

0.14.31.2.8 operator<() [8/11]

```
bool smtrat::cadcells::operators::properties::operator< (
    const poly_semi_sgn_inv & lhs,
    const poly_semi_sgn_inv & rhs ) [inline]
```

0.14.31.2.9 operator<() [9/11]

```
bool smtrat::cadcells::operators::properties::operator< (
    const poly_sgn_inv & lhs,
    const poly_sgn_inv & rhs ) [inline]
```

0.14.31.2.10 operator<() [10/11]

```
bool smtrat::cadcells::operators::properties::operator< (
    const root_ordering_holds & lhs,
    const root_ordering_holds & rhs ) [inline]
```

0.14.31.2.11 operator<() [11/11]

```
bool smtrat::cadcells::operators::properties::operator< (
    const root_well_def & lhs,
    const root_well_def & rhs ) [inline]
```

0.14.31.2.12 operator<<() [1/11]

```
std::ostream& smtrat::cadcells::operators::properties::operator<<
(
    std::ostream & os,
    const cell_connected & data ) [inline]
```

0.14.31.2.13 operator<<() [2/11]

```
std::ostream& smtrat::cadcells::operators::properties::operator<<
(
    std::ostream & os,
    const poly_additional_root_outside & data ) [inline]
```

0.14.31.2.14 operator<<() [3/11]

```
std::ostream& smtrat::cadcells::operators::properties::operator<<
(
    std::ostream & os,
    const poly_irreducible_semi_sgn_inv & data ) [inline]
```

0.14.31.2.15 operator<<() [4/11]

```
std::ostream& smtrat::cadcells::operators::properties::operator<<
(
    std::ostream & os,
    const poly_irreducible_sgn_inv & data ) [inline]
```

0.14.31.2.16 operator<<() [5/11]

```
std::ostream& smtrat::cadcells::operators::properties::operator<<
(
    std::ostream & os,
    const poly_ord_inv & data ) [inline]
```

0.14.31.2.17 operator<<() [6/11]

```
std::ostream& smtrat::cadcells::operators::properties::operator<<
(
    std::ostream & os,
    const poly_ord_inv_base & data ) [inline]
```

0.14.31.2.18 operator<<() [7/11]

```
std::ostream& smtrat::cadcells::operators::properties::operator<<
(
    std::ostream & os,
    const poly_pdel & data ) [inline]
```

0.14.31.2.19 operator<<() [8/11]

```
std::ostream& smtrat::cadcells::operators::properties::operator<<
(
    std::ostream & os,
    const poly_semi_sgn_inv & data ) [inline]
```

0.14.31.2.20 operator<<() [9/11]

```
std::ostream& smtrat::cadcells::operators::properties::operator<<
(
    std::ostream & os,
    const poly_sgn_inv & data ) [inline]
```

0.14.31.2.21 operator<<() [10/11]

```
std::ostream& smtrat::cadcells::operators::properties::operator<<
(
    std::ostream & os,
    const root_ordering_holds & data )  [inline]
```

0.14.31.2.22 operator<<() [11/11]

```
std::ostream& smtrat::cadcells::operators::properties::operator<<
(
    std::ostream & os,
    const root_well_def & data )  [inline]
```

0.14.31.2.23 operator==() [1/11]

```
bool smtrat::cadcells::operators::properties::operator== (
    const cell_connected & lhs,
    const cell_connected & rhs )  [inline]
```

0.14.31.2.24 operator==() [2/11]

```
bool smtrat::cadcells::operators::properties::operator== (
    const poly_additional_root_outside & lhs,
    const poly_additional_root_outside & rhs )  [inline]
```

0.14.31.2.25 operator==() [3/11]

```
bool smtrat::cadcells::operators::properties::operator== (
    const poly_irreducible_semi_sgn_inv & lhs,
    const poly_irreducible_semi_sgn_inv & rhs )  [inline]
```

0.14.31.2.26 operator==() [4/11]

```
bool smtrat::cadcells::operators::properties::operator== (
    const poly_irreducible_sgn_inv & lhs,
    const poly_irreducible_sgn_inv & rhs )  [inline]
```

0.14.31.2.27 operator==() [5/11]

```
bool smtrat::cadcells::operators::properties::operator== (
    const poly_ord_inv & lhs,
    const poly_ord_inv & rhs )  [inline]
```

0.14.31.2.28 operator==() [6/11]

```
bool smtrat::cadcells::operators::properties::operator== (
    const poly_ord_inv_base & lhs,
    const poly_ord_inv_base & rhs )  [inline]
```

0.14.31.2.29 operator==() [7/11]

```
bool smtrat::cadcells::operators::properties::operator== (
    const poly_pdel & lhs,
    const poly_pdel & rhs )  [inline]
```

0.14.31.2.30 operator==() [8/11]

```
bool smtrat::cadcells::operators::properties::operator== (
    const poly_semi_sgn_inv & lhs,
    const poly_semi_sgn_inv & rhs )  [inline]
```

0.14.31.2.31 operator==() [9/11]

```
bool smtrat::cadcells::operators::properties::operator== (
    const poly_sgn_inv & lhs,
    const poly_sgn_inv & rhs )  [inline]
```

0.14.31.2.32 operator==() [10/11]

```
bool smtrat::cadcells::operators::properties::operator== (
    const root_ordering_holds & lhs,
    const root_ordering_holds & rhs ) [inline]
```

0.14.31.2.33 operator==() [11/11]

```
bool smtrat::cadcells::operators::properties::operator== (
    const root_well_def & lhs,
    const root_well_def & rhs ) [inline]
```

0.14.32 smtrat::cadcells::operators::rules Namespace Reference

Implementation of derivation rules.

Namespaces

- additional_root_outside_util
- compound_util
- filter_util
- ordering_util

Functions

- template<typename P >
  void root_well_def (datastructures::SampledDerivation< P > &deriv, datastructures::IndexedRoot root)
- template<typename P >
  bool is_trivial_root_well_def (datastructures::SampledDerivation< P > &deriv, datastructures::IndexedRoot root)
- template<typename P >
  bool poly_non_null (datastructures::SampledDerivation< P > &deriv, datastructures::PolyRef poly)
- template<typename P >
  bool poly_pdel (datastructures::SampledDerivation< P > &deriv, datastructures::PolyRef poly)
- template<typename P >
  void poly_irreducible_ord_inv (datastructures::SampledDerivation< P > &deriv, datastructures::PolyRef poly)
- template<typename P >
  void poly_ord_inv (datastructures::SampledDerivation< P > &deriv, datastructures::PolyRef poly)
- template<typename P >
  void poly_ord_inv_base (datastructures::SampledDerivation< P > &deriv, const datastructures::SymbolicInterval &cell, const datastructures::IndexedRootOrdering &, datastructures::PolyRef poly)
- template<typename P >
  void poly_sgn_inv (datastructures::DelineatedDerivation< P > &deriv, datastructures::PolyRef poly)
- template<typename P >
  void poly_semi_sgn_inv (datastructures::DelineatedDerivation< P > &deriv, datastructures::PolyRef poly)
- template<typename P >
  void poly_irreducible_nonzero_sgn_inv (datastructures::DelineatedDerivation< P > &deriv, datastructures::PolyRef poly)
- template<typename P >
  void poly_irreducible_nonzero_semi_sgn_inv (datastructures::DelineatedDerivation< P > &deriv, datastructures::PolyRef poly)
- template<typename P >
  void delineate (datastructures::DelineatedDerivation< P > &deriv, const properties::poly_irreducible_sgn_inv &prop)
- template<typename P >
  void delineate (datastructures::DelineatedDerivation< P > &deriv, const properties::poly_irreducible_semi_sgn_inv &prop)
- template<typename P >
  void cell_connected (datastructures::SampledDerivation< P > &deriv, const datastructures::SymbolicInterval &cell, const datastructures::IndexedRootOrdering &ordering)

- template<typename P >
  void cell_analytic_submanifold ([[maybe_unused]] datastructures::SampledDerivation< P > &deriv, const datastructures::SymbolicInterval &)
- template<typename P >
  void poly_irreducible_sgn_inv_ec (datastructures::SampledDerivation< P > &deriv, const datastructures::SymbolicInterval &cell, datastructures::PolyRef poly)
- template<typename P >
  void poly_irreducible_semi_sgn_inv_ec (datastructures::SampledDerivation< P > &deriv, const datastructures::SymbolicInterva &cell, datastructures::PolyRef poly)
- template<typename P >
  void root_represents (datastructures::SampledDerivation< P > &deriv, datastructures::IndexedRoot root)
- template<typename P >
  void cell_represents (datastructures::SampledDerivation< P > &deriv, const datastructures::SymbolicInterval &cell)
- template<typename P >
  void root_ordering_holds (datastructures::SampledDerivation< P > &deriv, const datastructures::IndexedRootOrdering &ordering)
- template<typename P >
  void poly_additional_root_outside (datastructures::SampledDerivation< P > &deriv, const datastructures::SymbolicInterval &cell, const datastructures::IndexedRootOrdering &ordering, datastructures::PolyRef poly)
- template<typename P >
  void poly_irreducible_sgn_inv (datastructures::SampledDerivation< P > &deriv, const datastructures::SymbolicInterval &cell, const datastructures::IndexedRootOrdering &ordering, datastructures::PolyRef poly)
- template<typename P >
  void poly_irreducible_semi_sgn_inv (datastructures::SampledDerivation< P > &deriv, const datastructures::SymbolicInterval &cell, const datastructures::IndexedRootOrdering &ordering, datastructures::PolyRef poly)
- template<typename P >
  void poly_irreducible_sgn_inv_filtered (datastructures::SampledDerivation< P > &deriv, const datastructures::SymbolicInterval &, const datastructures::IndexedRootOrdering &, datastructures::PolyRef poly)
- template<typename P >
  void poly_irreducible_semi_sgn_inv_filtered (datastructures::SampledDerivation< P > &deriv, const datastructures::SymbolicInterval &, const datastructures::IndexedRootOrdering &, datastructures::PolyRef poly)
- template<typename P >
  void covering_holds (datastructures::DelineatedDerivation< P > &, const datastructures::CoveringDescription &covering, const datastructures::IndexedRootOrdering &ordering)
- template<typename P >
  void delineate_all_compound (datastructures::SampledDerivation< P > &deriv, const properties::root_ordering_holds &prop)
- template<typename P >
  void delineate_all (datastructures::SampledDerivation< P > &deriv, const properties::root_ordering_holds &prop)
- template<typename P >
  void delineate_samples (datastructures::SampledDerivation< P > &deriv, const properties::root_ordering_holds &prop)
- template<typename P >
  void delineate_all_selective (datastructures::SampledDerivation< P > &deriv, const properties::root_ordering_holds &prop)
- template<typename P >
  void delineate_bounds (datastructures::SampledDerivation< P > &deriv, const properties::root_ordering_holds &prop)
- template<typename P >
  void delineate_noop (datastructures::SampledDerivation< P > &deriv, const properties::root_ordering_holds &prop)
- template<typename P >
  bool root_ordering_holds_delineated (datastructures::SampledDerivation< P > &deriv, const datastructures::SymbolicInterval &, const datastructures::IndexedRootOrdering &underlying_ordering, const datastructures::IndexedRootOrdering &ordering)

0.14.32.1 Detailed Description

Implementation of derivation rules.
Each rule is realized by a function which works on a derivation object. The parameters of the respective properties are passed as function parameter. The function realizing a derivation rule might either call other derivation rules or add properties to the given derivation.

0.14.32.2 Function Documentation

0.14.32.2.1 cell_analytic_submanifold() template<typename P >

```
void smtrat::cadcells::operators::rules::cell_analytic_submanifold (
            [[maybe_unused] ] datastructures::SampledDerivation< P > & deriv,
            const datastructures::SymbolicInterval &  )
```

0.14.32.2.2 cell_connected() template<typename P >

```
void smtrat::cadcells::operators::rules::cell_connected (
            datastructures::SampledDerivation< P > & deriv,
            const datastructures::SymbolicInterval & cell,
            const datastructures::IndexedRootOrdering & ordering )
```

0.14.32.2.3 cell_represents() template<typename P >

```
void smtrat::cadcells::operators::rules::cell_represents (
            datastructures::SampledDerivation< P > & deriv,
            const datastructures::SymbolicInterval & cell )
```

0.14.32.2.4 covering_holds() template<typename P >

```
void smtrat::cadcells::operators::rules::covering_holds (
            datastructures::DelineatedDerivation< P > & ,
            const datastructures::CoveringDescription & covering,
            const datastructures::IndexedRootOrdering & ordering )
```

0.14.32.2.5 delineate() [1/2] template<typename P >

```
void smtrat::cadcells::operators::rules::delineate (
            datastructures::DelineatedDerivation< P > & deriv,
            const properties::poly_irreducible_semi_sgn_inv & prop )
```

0.14.32.2.6 delineate() [2/2] template<typename P >

```
void smtrat::cadcells::operators::rules::delineate (
            datastructures::DelineatedDerivation< P > & deriv,
            const properties::poly_irreducible_sgn_inv & prop )
```

0.14.32.2.7 delineate_all() template<typename P >

```
void smtrat::cadcells::operators::rules::delineate_all (
            datastructures::SampledDerivation< P > & deriv,
            const properties::root_ordering_holds & prop )
```

0.14.32.2.8 delineate_all_compound()

```
template<typename P >
void smtrat::cadcells::operators::rules::delineate_all_compound (
            datastructures::SampledDerivation< P > & deriv,
            const properties::root_ordering_holds & prop )
```

0.14.32.2.9 delineate_all_selective()

```
template<typename P >
void smtrat::cadcells::operators::rules::delineate_all_selective (
            datastructures::SampledDerivation< P > & deriv,
            const properties::root_ordering_holds & prop )
```

0.14.32.2.10 delineate_bounds()

```
template<typename P >
void smtrat::cadcells::operators::rules::delineate_bounds (
            datastructures::SampledDerivation< P > & deriv,
            const properties::root_ordering_holds & prop )
```

0.14.32.2.11 delineate_noop()

```
template<typename P >
void smtrat::cadcells::operators::rules::delineate_noop (
            datastructures::SampledDerivation< P > & deriv,
            const properties::root_ordering_holds & prop )
```

0.14.32.2.12 delineate_samples()

```
template<typename P >
void smtrat::cadcells::operators::rules::delineate_samples (
            datastructures::SampledDerivation< P > & deriv,
            const properties::root_ordering_holds & prop )
```

0.14.32.2.13 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 )
```

0.14.32.2.14 poly_additional_root_outside()

```
template<typename P >
void smtrat::cadcells::operators::rules::poly_additional_root_outside (
            datastructures::SampledDerivation< P > & deriv,
            const datastructures::SymbolicInterval & cell,
            const datastructures::IndexedRootOrdering & ordering,
            datastructures::PolyRef poly )
```

0.14.32.2.15 poly_irreducible_nonzero_semi_sgn_inv()

```
template<typename P >
void smtrat::cadcells::operators::rules::poly_irreducible_nonzero_semi_sgn_inv (
            datastructures::DelineatedDerivation< P > & deriv,
            datastructures::PolyRef poly )
```

0.14.32.2.16 poly_irreducible_nonzero_sgn_inv()

```
template<typename P >
void smtrat::cadcells::operators::rules::poly_irreducible_nonzero_sgn_inv (
            datastructures::DelineatedDerivation< P > & deriv,
            datastructures::PolyRef poly )
```

0.14.32.2.17 poly_irreducible_ord_inv()

```
template<typename P >
void smtrat::cadcells::operators::rules::poly_irreducible_ord_inv (
            datastructures::SampledDerivation< P > & deriv,
            datastructures::PolyRef poly )
```

0.14.32.2.18 poly_irreducible_semi_sgn_inv()

```
template<typename P >
void smtrat::cadcells::operators::rules::poly_irreducible_semi_sgn_inv (
            datastructures::SampledDerivation< P > & deriv,
            const datastructures::SymbolicInterval & cell,
            const datastructures::IndexedRootOrdering & ordering,
            datastructures::PolyRef poly )
```

0.14.32.2.19 poly_irreducible_semi_sgn_inv_ec()

```
template<typename P >
void smtrat::cadcells::operators::rules::poly_irreducible_semi_sgn_inv_ec (
            datastructures::SampledDerivation< P > & deriv,
            const datastructures::SymbolicInterval & cell,
            datastructures::PolyRef poly )
```

0.14.32.2.20 poly_irreducible_semi_sgn_inv_filtered()

```
template<typename P >
void smtrat::cadcells::operators::rules::poly_irreducible_semi_sgn_inv_filtered (
            datastructures::SampledDerivation< P > & deriv,
            const datastructures::SymbolicInterval & ,
            const datastructures::IndexedRootOrdering & ,
            datastructures::PolyRef poly )
```

0.14.32.2.21 poly_irreducible_sgn_inv()

```
template<typename P >
void smtrat::cadcells::operators::rules::poly_irreducible_sgn_inv (
            datastructures::SampledDerivation< P > & deriv,
            const datastructures::SymbolicInterval & cell,
            const datastructures::IndexedRootOrdering & ordering,
            datastructures::PolyRef poly )
```

0.14.32.2.22 poly_irreducible_sgn_inv_ec()

```
template<typename P >
void smtrat::cadcells::operators::rules::poly_irreducible_sgn_inv_ec (
            datastructures::SampledDerivation< P > & deriv,
            const datastructures::SymbolicInterval & cell,
            datastructures::PolyRef poly )
```

0.14.32.2.23 poly_irreducible_sgn_inv_filtered()

```
template<typename P >
void smtrat::cadcells::operators::rules::poly_irreducible_sgn_inv_filtered (
            datastructures::SampledDerivation< P > & deriv,
            const datastructures::SymbolicInterval & ,
            const datastructures::IndexedRootOrdering & ,
            datastructures::PolyRef poly )
```

0.14.32.2.24 poly_non_null()

```
template<typename P >
bool smtrat::cadcells::operators::rules::poly_non_null (
            datastructures::SampledDerivation< P > & deriv,
            datastructures::PolyRef poly )
```

0.14.32.2.25 poly_ord_inv()

```
template<typename P >
void smtrat::cadcells::operators::rules::poly_ord_inv (
            datastructures::SampledDerivation< P > & deriv,
            datastructures::PolyRef poly )
```

0.14.32.2.26 poly_ord_inv_base()

```
template<typename P >
void smtrat::cadcells::operators::rules::poly_ord_inv_base (
            datastructures::SampledDerivation< P > & deriv,
            const datastructures::SymbolicInterval & cell,
            const datastructures::IndexedRootOrdering & ,
            datastructures::PolyRef poly )
```

0.14.32.2.27 poly_pdel()

```
template<typename P >
bool smtrat::cadcells::operators::rules::poly_pdel (
            datastructures::SampledDerivation< P > & deriv,
            datastructures::PolyRef poly )
```

0.14.32.2.28 poly_semi_sgn_inv()

```
template<typename P >
void smtrat::cadcells::operators::rules::poly_semi_sgn_inv (
            datastructures::DelineatedDerivation< P > & deriv,
            datastructures::PolyRef poly )
```

0.14.32.2.29 poly_sgn_inv()

```
template<typename P >
void smtrat::cadcells::operators::rules::poly_sgn_inv (
            datastructures::DelineatedDerivation< P > & deriv,
            datastructures::PolyRef poly )
```

0.14.32.2.30 root_ordering_holds()

```
template<typename P >
void smtrat::cadcells::operators::rules::root_ordering_holds (
            datastructures::SampledDerivation< P > & deriv,
            const datastructures::IndexedRootOrdering & ordering )
```

0.14.32.2.31 root_ordering_holds_delineated()

```
template<typename P >
bool smtrat::cadcells::operators::rules::root_ordering_holds_delineated (
            datastructures::SampledDerivation< P > & deriv,
            const datastructures::SymbolicInterval & ,
            const datastructures::IndexedRootOrdering & underlying_ordering,
            const datastructures::IndexedRootOrdering & ordering )
```

0.14.32.2.32 root_represents()

```
template<typename P >
void smtrat::cadcells::operators::rules::root_represents (
            datastructures::SampledDerivation< P > & deriv,
            datastructures::IndexedRoot root )
```

0.14.32.2.33 root_well_def()

```
template<typename P >
void smtrat::cadcells::operators::rules::root_well_def (
            datastructures::SampledDerivation< P > & deriv,
            datastructures::IndexedRoot root )
```

0.14.33 smtrat::cadcells::operators::rules::additional_root_outside_util Namespace Reference

Functions

- boost::container::flat_set< datastructures::PolyRef > resultant_polys (const datastructures::PolyRef &poly, const datastructures::IndexedRootOrdering &ordering)
- template<typename P >
  std::optional< datastructures::IndexedRoot > protect_lower (datastructures::SampledDerivation< P > &deriv, const datastructures::SymbolicInterval &cell, const datastructures::IndexedRootOrdering &ordering, const datastructures::PolyRef &poly, const boost::container::flat_set< datastructures::PolyRef > &res_polys, bool strict=false)
- template<typename P >
  std::optional< datastructures::IndexedRoot > protect_upper (datastructures::SampledDerivation< P > &deriv, const datastructures::SymbolicInterval &cell, const datastructures::IndexedRootOrdering &ordering, const datastructures::PolyRef &poly, const boost::container::flat_set< datastructures::PolyRef > &res_polys, bool strict=false)

0.14.33.1 Function Documentation

0.14.33.1.1 protect_lower()

```
template<typename P >
std::optional<datastructures::IndexedRoot> smtrat::cadcells::operators::rules::additional_↩
root_outside_util::protect_lower (
            datastructures::SampledDerivation< P > & deriv,
            const datastructures::SymbolicInterval & cell,
            const datastructures::IndexedRootOrdering & ordering,
            const datastructures::PolyRef & poly,
            const boost::container::flat_set< datastructures::PolyRef > & res_polys,
            bool strict = false )  [inline]
```

0.14.33.1.2 protect_upper()

```
template<typename P >
std::optional<datastructures::IndexedRoot> smtrat::cadcells::operators::rules::additional_↩
root_outside_util::protect_upper (
            datastructures::SampledDerivation< P > & deriv,
            const datastructures::SymbolicInterval & cell,
            const datastructures::IndexedRootOrdering & ordering,
            const datastructures::PolyRef & poly,
            const boost::container::flat_set< datastructures::PolyRef > & res_polys,
            bool strict = false )  [inline]
```

0.14.33.1.3 resultant_polys()

```
boost::container::flat_set<datastructures::PolyRef> smtrat::cadcells↩
::operators::rules::additional_root_outside_util::resultant_polys (
            const datastructures::PolyRef & poly,
            const datastructures::IndexedRootOrdering & ordering )  [inline]
```

0.14.34 smtrat::cadcells::operators::rules::compound_util Namespace Reference

Functions

- std::pair< RAN, std::vector< datastructures::IndexedRoot > > evaluate_min (datastructures::Projections &proj, const Assignment &ass, const std::vector< datastructures::IndexedRoot > &roots)
- std::pair< RAN, std::vector< datastructures::IndexedRoot > > evaluate_max (datastructures::Projections &proj, const Assignment &ass, const std::vector< datastructures::IndexedRoot > &roots)
- std::pair< RAN, std::vector< datastructures::IndexedRoot > > evaluate (datastructures::Projections &proj, const Assignment &ass, const datastructures::CompoundMinMax &bound)

- std::pair< RAN, std::vector< datastructures::IndexedRoot > > evaluate (datastructures::Projections &proj, const Assignment &ass, const datastructures::CompoundMaxMin &bound)

0.14.34.1 Function Documentation

0.14.34.1.1 evaluate() [1/2]

```
std::pair<RAN,std::vector<datastructures::IndexedRoot> > smtrat↩
::cadcells::operators::rules::compound_util::evaluate (
            datastructures::Projections & proj,
            const Assignment & ass,
            const datastructures::CompoundMaxMin & bound ) [inline]
```

0.14.34.1.2 evaluate() [2/2]

```
std::pair<RAN,std::vector<datastructures::IndexedRoot> > smtrat↩
::cadcells::operators::rules::compound_util::evaluate (
            datastructures::Projections & proj,
            const Assignment & ass,
            const datastructures::CompoundMinMax & bound ) [inline]
```

0.14.34.1.3 evaluate_max()

```
std::pair<RAN,std::vector<datastructures::IndexedRoot> > smtrat↩
::cadcells::operators::rules::compound_util::evaluate_max (
            datastructures::Projections & proj,
            const Assignment & ass,
            const std::vector< datastructures::IndexedRoot > & roots ) [inline]
```

0.14.34.1.4 evaluate_min()

```
std::pair<RAN,std::vector<datastructures::IndexedRoot> > smtrat↩
::cadcells::operators::rules::compound_util::evaluate_min (
            datastructures::Projections & proj,
            const Assignment & ass,
            const std::vector< datastructures::IndexedRoot > & roots ) [inline]
```

0.14.35 smtrat::cadcells::operators::rules::filter_util Namespace Reference

Enumerations

- enum class result {
  NORMAL , INCLUSIVE , NORMAL_OPTIONAL , INCLUSIVE_OPTIONAL ,
  OMIT }

Functions

- template<typename P >
  void pseudo_order_invariant (datastructures::SampledDerivation< P > &deriv, const datastructures::PolyRef poly, const boost::container::flat_set< datastructures::PolyRef > &considered_polys)
- template<typename P >
  std::optional< carl::Interval< RAN > > delineable_interval (datastructures::Projections &proj, const Assignment &sample, const boost::container::flat_set< datastructures::PolyRef > &polys)
- template<typename P >
  void filter_roots (datastructures::DelineatedDerivation< P > &deriv, const datastructures::PolyRef poly, std↩::function< result(const RAN &)> filter_condition)
- template<typename P >
  auto projection_root (const datastructures::DelineatedDerivation< P > &deriv, const RAN &root)
- bool has_common_real_root (datastructures::Projections &proj, Assignment ass, const datastructures::PolyRef &poly1, const datastructures::PolyRef &poly2)

0.14.35.1 Enumeration Type Documentation

0.14.35.1.1 result `enum smtrat::cadcells::operators::rules::filter_util::result [strong]`

Enumerator

NORMAL	
INCLUSIVE	
NORMAL_OPTIONAL	
INCLUSIVE_OPTIONAL	
OMIT	

0.14.35.2 Function Documentation

0.14.35.2.1 delineable_interval()

```
template<typename P >
std::optional<carl::Interval<RAN> > smtrat::cadcells::operators::rules::filter_util::delineable↩
_interval (
            datastructures::Projections & proj,
            const Assignment & sample,
            const boost::container::flat_set< datastructures::PolyRef > & polys ) [inline]
```

0.14.35.2.2 filter_roots()

```
template<typename P >
void smtrat::cadcells::operators::rules::filter_util::filter_roots (
            datastructures::DelineatedDerivation< P > & deriv,
            const datastructures::PolyRef poly,
            std::function< result(const RAN &)> filter_condition ) [inline]
```

0.14.35.2.3 has_common_real_root()

```
bool smtrat::cadcells::operators::rules::filter_util::has↩
_common_real_root (
            datastructures::Projections & proj,
            Assignment ass,
            const datastructures::PolyRef & poly1,
            const datastructures::PolyRef & poly2 ) [inline]
```

0.14.35.2.4 projection_root()

```
template<typename P >
auto smtrat::cadcells::operators::rules::filter_util::projection_root (
            const datastructures::DelineatedDerivation< P > & deriv,
            const RAN & root ) [inline]
```

0.14.35.2.5 pseudo_order_invariant()

```
template<typename P >
void smtrat::cadcells::operators::rules::filter_util::pseudo_order_invariant (
            datastructures::SampledDerivation< P > & deriv,
            const datastructures::PolyRef poly,
            const boost::container::flat_set< datastructures::PolyRef > & considered_polys )
[inline]
```

0.14.36 smtrat::cadcells::operators::rules::ordering_util Namespace Reference

Typedefs

- using Decomposition = boost::container::flat_map< std::pair< datastructures::PolyRef, datastructures::PolyRef >, std::vector< datastructures::IndexedRootRelation > >

Functions

- void add_to_decomposition (Decomposition &result, datastructures::PolyRef p1, datastructures::PolyRef p2, const datastructures::IndexedRootRelation &rel)
- auto decompose (const datastructures::IndexedRootOrdering &ordering)

0.14.36.1 Typedef Documentation

0.14.36.1.1 Decomposition

```
using smtrat::cadcells::operators::rules::ordering_util::Decomposition
= typedef boost::container::flat_map<std::pair<datastructures::PolyRef,datastructures::PolyRef>,std↩
::vector<datastructures::IndexedRootRelation> >
```

0.14.36.2 Function Documentation

0.14.36.2.1 add_to_decomposition()

```
void smtrat::cadcells::operators::rules::ordering_util↩
::add_to_decomposition (
            Decomposition & result,
            datastructures::PolyRef p1,
            datastructures::PolyRef p2,
            const datastructures::IndexedRootRelation & rel )  [inline]
```

0.14.36.2.2 decompose()

```
auto smtrat::cadcells::operators::rules::ordering_util::decompose (
            const datastructures::IndexedRootOrdering & ordering )  [inline]
```

0.14.37 smtrat::cadcells::representation Namespace Reference

Heuristics for computing representations.

Namespaces

- approximation
- util

Data Structures

- struct cell

  Note: If connected(i) holds, then the indexed root ordering must contain an ordering between the interval bounds.
- struct covering
- struct cell< CellHeuristic::BIGGEST_CELL_APPROXIMATION >
- struct cell< CellHeuristic::BIGGEST_CELL >
- struct cell< CellHeuristic::BIGGEST_CELL_EW >
- struct cell< CellHeuristic::CHAIN_EQ >
- struct cell< CellHeuristic::LOWEST_DEGREE_BARRIERS_EQ >
- struct cell< CellHeuristic::LOWEST_DEGREE_BARRIERS >
- struct cell< CellHeuristic::LOWEST_DEGREE_BARRIERS_EW >
- struct covering< CoveringHeuristic::BIGGEST_CELL_COVERING >

- struct covering< CoveringHeuristic::BIGGEST_CELL_COVERING_EW >
- struct covering< CoveringHeuristic::BIGGEST_CELL_COVERING_MIN_TDEG >
- struct covering< CoveringHeuristic::CHAIN_COVERING >

Typedefs

- using IR = datastructures::IndexedRoot

Enumerations

- enum CellHeuristic { BIGGEST_CELL , CHAIN_EQ , LOWEST_DEGREE_BARRIERS , LOWEST_DEGREE_BARRIERS_EQ , BIGGEST_CELL_EW , LOWEST_DEGREE_BARRIERS_EW , BIGGEST_CELL_APPROXIMATION }
- enum CoveringHeuristic { BIGGEST_CELL_COVERING , CHAIN_COVERING , BIGGEST_CELL_COVERING_EW , BIGGEST_CELL_COVERING_MIN_TDEG }

Functions

- std::ostream & operator<< (std::ostream &os, CellHeuristic heuristic)
- std::ostream & operator<< (std::ostream &os, CoveringHeuristic heuristic)
- template<typename T > void compute_section_all_equational (datastructures::SampledDerivationRef< T > &der, datastructures::CellRepresentation< T > &response)
- template<typename T > void maintain_connectedness (datastructures::SampledDerivationRef< T > &der, datastructures::CellRepresentation< T > &response, bool enable_weak=false)
- template<typename T > void compute_barriers (datastructures::SampledDerivationRef< T > &der, datastructures::CellRepresentation< T > &response, bool section, bool enable_weak=false)
- template<typename T > std::vector< datastructures::SampledDerivationRef< T > > compute_min_derivs (const std::vector< datastructures::SampledDerivationRef< T >> &derivs)
- template<typename T > datastructures::IndexedRootOrdering compute_default_ordering (const std::vector< datastructures::CellRepresentation< T >> &cells, bool enable_weak=false)

Variables

- static const char * CellHeuristicStrings [ ] = { "BIGGEST_CELL", "CHAIN_EQ", "LOWEST_DEGREE_BARRIERS", "LOWEST_DEGREE_BARRIERS_EQ", "BIGGEST_CELL_EW", "LOWEST_DEGREE_BARRIERS_EW" }
- static const char * CoveringHeuristicStrings [ ] = { "BIGGEST_CELL_COVERING", "CHAIN_COVERING", "BIGGEST_CELL_COVERING_EW", "BIGGEST_CELL_COVERING_MIN_TDEG" }

0.14.37.1 Detailed Description

Heuristics for computing representations.

0.14.37.2 Typedef Documentation

0.14.37.2.1 IR `using smtrat::cadcells::representation::IR = typedef datastructures::IndexedRoot`

0.14.37.3 Enumeration Type Documentation

0.14.37.3.1 CellHeuristic `enum smtrat::cadcells::representation::CellHeuristic`

Enumerator

BIGGEST_CELL	
CHAIN_EQ	
LOWEST_DEGREE_BARRIERS	
LOWEST_DEGREE_BARRIERS_EQ	
BIGGEST_CELL_EW	
LOWEST_DEGREE_BARRIERS_EW	
BIGGEST_CELL_APPROXIMATION	

0.14.37.3.2 CoveringHeuristic `enum smtrat::cadcells::representation::CoveringHeuristic`

Enumerator

BIGGEST_CELL_COVERING	
CHAIN_COVERING	
BIGGEST_CELL_COVERING_EW	
BIGGEST_CELL_COVERING_MIN_TDEG	

0.14.37.4 Function Documentation

0.14.37.4.1 compute_barriers() `template<typename T >`

```
void smtrat::cadcells::representation::compute_barriers (
            datastructures::SampledDerivationRef< T > & der,
            datastructures::CellRepresentation< T > & response,
            bool section,
            bool enable_weak = false )  [inline]
```

0.14.37.4.2 compute_default_ordering() `template<typename T >`

```
datastructures::IndexedRootOrdering smtrat::cadcells::representation::compute_default_ordering
(
            const std::vector< datastructures::CellRepresentation< T >> & cells,
            bool enable_weak = false )
```

0.14.37.4.3 compute_min_derivs() `template<typename T >`

```
std::vector<datastructures::SampledDerivationRef<T> > smtrat::cadcells::representation↩
::compute_min_derivs (
            const std::vector< datastructures::SampledDerivationRef< T >> & derivs )
```

0.14.37.4.4 compute_section_all_equational() `template<typename T >`

```
void smtrat::cadcells::representation::compute_section_all_equational (
            datastructures::SampledDerivationRef< T > & der,
            datastructures::CellRepresentation< T > & response )  [inline]
```

0.14.37.4.5 maintain_connectedness() `template<typename T >`

```
void smtrat::cadcells::representation::maintain_connectedness (
```

```
datastructures::SampledDerivationRef< T > & der,
datastructures::CellRepresentation< T > & response,
bool enable_weak = false )
```

0.14.37.4.6 operator<<() [1/2]

```
std::ostream& smtrat::cadcells::representation::operator<< (
    std::ostream & os,
    CellHeuristic heuristic ) [inline]
```

0.14.37.4.7 operator<<() [2/2]

```
std::ostream& smtrat::cadcells::representation::operator<< (
    std::ostream & os,
    CoveringHeuristic heuristic ) [inline]
```

0.14.37.5 Variable Documentation

0.14.37.5.1 CellHeuristicStrings

```
const char* smtrat::cadcells::representation::CellHeuristic↩
Strings[] = { "BIGGEST_CELL", "CHAIN_EQ", "LOWEST_DEGREE_BARRIERS", "LOWEST_DEGREE_BARRIERS_EQ",
"BIGGEST_CELL_EW", "LOWEST_DEGREE_BARRIERS_EW" } [static]
```

0.14.37.5.2 CoveringHeuristicStrings

```
const char* smtrat::cadcells::representation::Covering↩
HeuristicStrings[] = { "BIGGEST_CELL_COVERING", "CHAIN_COVERING", "BIGGEST_CELL_COVERING_EW",
"BIGGEST_CELL_COVERING_MIN_TDEG" } [static]
```

0.14.38 smtrat::cadcells::representation::approximation Namespace Reference

Data Structures

- struct ApxSettings
- class CellApproximator
- class ApxCriteria

Typedefs

- using IR = datastructures::IndexedRoot

Enumerations

- enum ApxPoly {
  SIMPLE , LINEAR_GRADIENT , TAYLOR , TAYLOR_LIN ,
  MAXIMIZE }
- enum ApxRoot { SAMPLE_MID , SIMPLE_REPRESENTATION , STERN_BROCOT , FIXED_RATIO }

Functions

- const ApxSettings & apx_settings ()
- template<> IR CellApproximator::apx_bound< ApxPoly::SIMPLE > (const IR &, const RAN &bound, bool below)
- template<> IR CellApproximator::apx_bound< ApxPoly::LINEAR_GRADIENT > (const IR &p, const RAN &bound, bool below)
- template<> IR CellApproximator::apx_bound< ApxPoly::TAYLOR > (const IR &p, const RAN &bound, bool below)
- template<> IR CellApproximator::apx_bound< ApxPoly::TAYLOR_LIN > (const IR &p, const RAN &bound, bool below)

- template<> IR CellApproximator::apx_bound< ApxPoly::MAXIMIZE > (const IR &p, const RAN &bound, bool below)
- template<> IR CellApproximator::apx_between< ApxPoly::SIMPLE > (const IR &, const IR &, const RAN &l, const RAN &u)
- Rational mediant (Rational a, Rational b)
- Rational approximate_RAN (const RAN &r)
- Rational approximate_RAN_sb (const RAN &r)
- Rational approximate_RAN_below (const RAN &r)
- Rational approximate_RAN_above (const RAN &r)
- template<ApxRoot AR>
  Rational approximate_root_above (const RAN &inner, const RAN &outer)
- template<ApxRoot AR>
  Rational approximate_root_below (const RAN &inner, const RAN &outer)
- template<> Rational approximate_root_above< ApxRoot::SAMPLE_MID > (const RAN &inner, const RAN &outer)
- template<> Rational approximate_root_below< ApxRoot::SAMPLE_MID > (const RAN &inner, const RAN &outer)
- template<> Rational approximate_root_above< ApxRoot::SIMPLE_REPRESENTATION > (const RAN &inner, const RAN &outer)
- template<> Rational approximate_root_below< ApxRoot::SIMPLE_REPRESENTATION > (const RAN &inner, const RAN &outer)
- template<> Rational approximate_root_above< ApxRoot::STERN_BROCOT > (const RAN &inner, const RAN &outer)
- template<> Rational approximate_root_below< ApxRoot::STERN_BROCOT > (const RAN &inner, const RAN &outer)
- template<> Rational approximate_root_above< ApxRoot::FIXED_RATIO > (const RAN &inner, const RAN &outer)
- template<> Rational approximate_root_below< ApxRoot::FIXED_RATIO > (const RAN &inner, const RAN &outer)
- template<ApxRoot AR>
  Rational approximate_root (const RAN &inner, const RAN &outer, bool below)

0.14.38.1 Typedef Documentation

0.14.38.1.1 IR `typedef datastructures::IndexedRoot smtrat::cadcells::representation::approximation::IR`

0.14.38.2 Enumeration Type Documentation

0.14.38.2.1 ApxPoly `enum smtrat::cadcells::representation::approximation::ApxPoly`

Enumerator

SIMPLE	
LINEAR_GRADIENT	
TAYLOR	
TAYLOR_LIN	
MAXIMIZE	

0.14.38.2.2 ApxRoot `enum smtrat::cadcells::representation::approximation::ApxRoot`

Enumerator

SAMPLE_MID	
SIMPLE_REPRESENTATION	
STERN_BROCOT	
FIXED_RATIO	

0.14.38.3 Function Documentation

0.14.38.3.1 approximate_RAN()

```
Rational smtrat::cadcells::representation::approximation::approximate↩
_RAN (
            const RAN & r )  [inline]
```

0.14.38.3.2 approximate_RAN_above()

```
Rational smtrat::cadcells::representation::approximation↩
::approximate_RAN_above (
            const RAN & r )  [inline]
```

0.14.38.3.3 approximate_RAN_below()

```
Rational smtrat::cadcells::representation::approximation↩
::approximate_RAN_below (
            const RAN & r )  [inline]
```

0.14.38.3.4 approximate_RAN_sb()

```
Rational smtrat::cadcells::representation::approximation↩
::approximate_RAN_sb (
            const RAN & r )  [inline]
```

0.14.38.3.5 approximate_root()

```
template<ApxRoot AR>
Rational smtrat::cadcells::representation::approximation::approximate_root (
            const RAN & inner,
            const RAN & outer,
            bool below )  [inline]
```

0.14.38.3.6 approximate_root_above()

```
template<ApxRoot AR>
Rational smtrat::cadcells::representation::approximation::approximate_root_above (
            const RAN & inner,
            const RAN & outer )  [inline]
```

0.14.38.3.7 approximate_root_above< ApxRoot::FIXED_RATIO >()

```
template<>
Rational smtrat::cadcells::representation::approximation::approximate_root_above< ApxRoot::↩
FIXED_RATIO > (
            const RAN & inner,
            const RAN & outer )  [inline]
```

0.14.38.3.8 approximate_root_above< ApxRoot::SAMPLE_MID >()

```
template<>
Rational smtrat::cadcells::representation::approximation::approximate_root_above< ApxRoot::↩
SAMPLE_MID > (
```

```
            const RAN & inner,
            const RAN & outer )  [inline]
```

0.14.38.3.9 approximate_root_above< ApxRoot::SIMPLE_REPRESENTATION >() template<>

```
Rational smtrat::cadcells::representation::approximation::approximate_root_above< ApxRoot::↩
SIMPLE_REPRESENTATION > (
            const RAN & inner,
            const RAN & outer )  [inline]
```

0.14.38.3.10 approximate_root_above< ApxRoot::STERN_BROCOT >() template<>

```
Rational smtrat::cadcells::representation::approximation::approximate_root_above< ApxRoot::↩
STERN_BROCOT > (
            const RAN & inner,
            const RAN & outer )  [inline]
```

0.14.38.3.11 approximate_root_below() template<ApxRoot AR>

```
Rational smtrat::cadcells::representation::approximation::approximate_root_below (
            const RAN & inner,
            const RAN & outer )
```

0.14.38.3.12 approximate_root_below< ApxRoot::FIXED_RATIO >() template<>

```
Rational smtrat::cadcells::representation::approximation::approximate_root_below< ApxRoot::↩
FIXED_RATIO > (
            const RAN & inner,
            const RAN & outer )  [inline]
```

0.14.38.3.13 approximate_root_below< ApxRoot::SAMPLE_MID >() template<>

```
Rational smtrat::cadcells::representation::approximation::approximate_root_below< ApxRoot::↩
SAMPLE_MID > (
            const RAN & inner,
            const RAN & outer )  [inline]
```

0.14.38.3.14 approximate_root_below< ApxRoot::SIMPLE_REPRESENTATION >() template<>

```
Rational smtrat::cadcells::representation::approximation::approximate_root_below< ApxRoot::↩
SIMPLE_REPRESENTATION > (
            const RAN & inner,
            const RAN & outer )  [inline]
```

0.14.38.3.15 approximate_root_below< ApxRoot::STERN_BROCOT >() template<>

```
Rational smtrat::cadcells::representation::approximation::approximate_root_below< ApxRoot::↩
STERN_BROCOT > (
            const RAN & inner,
            const RAN & outer )  [inline]
```

0.14.38.3.16 apx_settings() const ApxSettings& smtrat::cadcells::representation::approximation↩ ::apx_settings ( ) [inline]

0.14.38.3.17 CellApproximator::apx_between< ApxPoly::SIMPLE >() template<>

```
IR smtrat::cadcells::representation::approximation::CellApproximator::apx_between< ApxPoly::↩
SIMPLE > (
            const IR & ,
            const IR & ,
            const RAN & l,
            const RAN & u )  [inline]
```

0.14.38.3.18 CellApproximator::apx_bound< ApxPoly::LINEAR_GRADIENT >() template<>

```
IR smtrat::cadcells::representation::approximation::CellApproximator::apx_bound< ApxPoly::↩
LINEAR_GRADIENT > (
            const IR & p,
            const RAN & bound,
            bool below )  [inline]
```

0.14.38.3.19 CellApproximator::apx_bound< ApxPoly::MAXIMIZE >() template<>

```
IR smtrat::cadcells::representation::approximation::CellApproximator::apx_bound< ApxPoly::↩
MAXIMIZE > (
            const IR & p,
            const RAN & bound,
            bool below )  [inline]
```

0.14.38.3.20 CellApproximator::apx_bound< ApxPoly::SIMPLE >() template<>

```
IR smtrat::cadcells::representation::approximation::CellApproximator::apx_bound< ApxPoly::↩
SIMPLE > (
            const IR & ,
            const RAN & bound,
            bool below )  [inline]
```

0.14.38.3.21 CellApproximator::apx_bound< ApxPoly::TAYLOR >() template<>

```
IR smtrat::cadcells::representation::approximation::CellApproximator::apx_bound< ApxPoly::↩
TAYLOR > (
            const IR & p,
            const RAN & bound,
            bool below )  [inline]
```

0.14.38.3.22 CellApproximator::apx_bound< ApxPoly::TAYLOR_LIN >() template<>

```
IR smtrat::cadcells::representation::approximation::CellApproximator::apx_bound< ApxPoly::↩
TAYLOR_LIN > (
            const IR & p,
            const RAN & bound,
            bool below )  [inline]
```

0.14.38.3.23 mediant()

```
Rational smtrat::cadcells::representation::approximation::mediant (
            Rational a,
            Rational b )  [inline]
```

0.14.39 smtrat::cadcells::representation::util Namespace Reference

Data Structures

- struct IntervalCompare

- struct PolyDelineation
- struct PolyDelineations

Typedefs

- template<typename T >
  using DerivationSet = boost::container::flat_set< datastructures::SampledDerivationRef< T >, IntervalCompare< T > >

Functions

- template<typename T >
  bool is_covering (const DerivationSet< T > &set)
- bool compare_simplest (datastructures::Projections &proj, datastructures::PolyRef p1, datastructures::PolyRef p2)
- bool max_degree (datastructures::Projections &proj, datastructures::RootFunction rf)
- bool compare_simplest (datastructures::Projections &proj, datastructures::RootFunction rf1, datastructures::RootFunction rf2)
- std::optional< datastructures::IndexedRoot > simplest_bound (datastructures::Projections &proj, const std::vector< datastructures::TaggedIndexedRoot > &bounds, const boost::container::flat_set< datastructures::PolyRef > &ignoring, bool enable_weak=false)
- datastructures::IndexedRoot simplest_bound (datastructures::Projections &proj, const std::vector< datastructures::TaggedIndexedRoot > &bounds, bool enable_weak=false)
- datastructures::SymbolicInterval compute_simplest_cell (datastructures::Projections &proj, const datastructures::DelineationInt &del, bool enable_weak=false)
- std::optional< datastructures::IndexedRootOrdering > simplest_biggest_cell_ordering (datastructures::Projections &, datastructures::Delineation &delin, datastructures::DelineationInterval &delin_interval, const datastructures::SymbolicInterva &interval, bool enable_weak=false)
- std::optional< datastructures::IndexedRootOrdering > simplest_chain_ordering (datastructures::Projections &proj, datastructures::Delineation &delin, bool enable_weak=false)
- void decompose (const datastructures::Delineation &delin, const datastructures::DelineationInterval &delin_interval, datastructures::Delineation &delin_out, PolyDelineations &poly_delin_out)
- void add_chain_ordering (datastructures::IndexedRootOrdering &out, const datastructures::PolyRef &poly, const PolyDelineation &poly_delin)
- void add_biggest_cell_ordering (datastructures::IndexedRootOrdering &out, const datastructures::PolyRef &poly, const PolyDelineation &poly_delin)
- void add_weird_ordering (datastructures::IndexedRootOrdering &out, const datastructures::Delineation &delin, const datastructures::DelineationInterval &delin_interval, const datastructures::SymbolicInterval &interval)

0.14.39.1 Typedef Documentation

0.14.39.1.1 DerivationSet `template<typename T >`

```
using smtrat::cadcells::representation::util::DerivationSet = typedef boost::container::flat_set<datastructures::SampledDerivationRef<T>, IntervalCompare<T> >
```

0.14.39.2 Function Documentation

0.14.39.2.1 add_biggest_cell_ordering()

```
void smtrat::cadcells::representation::util::add_biggest_cell_ordering (
            datastructures::IndexedRootOrdering & out,
            const datastructures::PolyRef & poly,
            const PolyDelineation & poly_delin )  [inline]
```

0.14.39.2.2 add_chain_ordering()

```
void smtrat::cadcells::representation::util::add_chain_↩
ordering (
            datastructures::IndexedRootOrdering & out,
            const datastructures::PolyRef & poly,
            const PolyDelineation & poly_delin )  [inline]
```

0.14.39.2.3 add_weird_ordering()

```
void smtrat::cadcells::representation::util::add_weird_↩
ordering (
            datastructures::IndexedRootOrdering & out,
            const datastructures::Delineation & delin,
            const datastructures::DelineationInterval & delin_interval,
            const datastructures::SymbolicInterval & interval )  [inline]
```

0.14.39.2.4 compare_simplest() [1/2]

```
bool smtrat::cadcells::representation::util::compare_↩
simplest (
            datastructures::Projections & proj,
            datastructures::PolyRef p1,
            datastructures::PolyRef p2 )  [inline]
```

0.14.39.2.5 compare_simplest() [2/2]

```
bool smtrat::cadcells::representation::util::compare_↩
simplest (
            datastructures::Projections & proj,
            datastructures::RootFunction rf1,
            datastructures::RootFunction rf2 )  [inline]
```

0.14.39.2.6 compute_simplest_cell()

```
datastructures::SymbolicInterval smtrat::cadcells::representation↩
::util::compute_simplest_cell (
            datastructures::Projections & proj,
            const datastructures::DelineationInterval & del,
            bool enable_weak = false )  [inline]
```

0.14.39.2.7 decompose()

```
void smtrat::cadcells::representation::util::decompose (
            const datastructures::Delineation & delin,
            const datastructures::DelineationInterval & delin_interval,
            datastructures::Delineation & delin_out,
            PolyDelineations & poly_delin_out )  [inline]
```

0.14.39.2.8 is_covering()

```
template<typename T >
bool smtrat::cadcells::representation::util::is_covering (
            const DerivationSet< T > & set )
```

0.14.39.2.9 max_degree()

```
bool smtrat::cadcells::representation::util::max_degree (
            datastructures::Projections & proj,
            datastructures::RootFunction rf )  [inline]
```

0.14.39.2.10 simplest_biggest_cell_ordering()

```
std::optional<datastructures::IndexedRootOrdering>
smtrat::cadcells::representation::util::simplest_biggest_cell_ordering (
            datastructures::Projections & ,
```

```
datastructures::Delineation & delin,
datastructures::DelineationInterval & delin_interval,
const datastructures::SymbolicInterval & interval,
bool enable_weak = false )  [inline]
```

0.14.39.2.11 simplest_bound() [1/2]

```
datastructures::IndexedRoot smtrat::cadcells::representation↩
::util::simplest_bound (
    datastructures::Projections & proj,
    const std::vector< datastructures::TaggedIndexedRoot > & bounds,
    bool enable_weak = false )  [inline]
```

0.14.39.2.12 simplest_bound() [2/2]

```
std::optional<datastructures::IndexedRoot> smtrat::cadcells↩
::representation::util::simplest_bound (
    datastructures::Projections & proj,
    const std::vector< datastructures::TaggedIndexedRoot > & bounds,
    const boost::container::flat_set< datastructures::PolyRef > & ignoring,
    bool enable_weak = false )  [inline]
```

0.14.39.2.13 simplest_chain_ordering()

```
std::optional<datastructures::IndexedRootOrdering> smtrat↩
::cadcells::representation::util::simplest_chain_ordering (
    datastructures::Projections & proj,
    datastructures::Delineation & delin,
    bool enable_weak = false )  [inline]
```

0.14.40 smtrat::compile_information Namespace Reference

Compile time generated information about compiler and system version.

Variables

- const std::string SystemName = "Linux"
- const std::string SystemVersion = "6.0.0-0.deb11.6-amd64"
- const std::string BuildType = "DEBUG"
- const std::string CXXCompiler = "/usr/bin/c++"
- const std::string CXXCompilerVersion = "11.3.0"
- const std::string GitRevisionSHA1 = "1bc26d0e6dfae377e44a45002efe7a18ef99e402"
- const std::string PackageName = "smtrat"
- const std::string ProjectName = "SMT-RAT"
- const std::string Version = "22.12"
- const std::string Website = "https://ths-rwth.github.io/smtrat/"
- const std::string GitVersion = "22.12"

0.14.40.1 Detailed Description

Compile time generated information about compiler and system version.

0.14.40.2 Variable Documentation

0.14.40.2.1 BuildType

```
const std::string smtrat::compile_information::BuildType = "DEBUG"
```

0.14.40.2.2 CXXCompiler

`const std::string smtrat::compile_information::CXXCompiler = "/usr/bin/c++"`

0.14.40.2.3 CXXCompilerVersion

`const std::string smtrat::compile_information::CXXCompiler↩ Version = "11.3.0"`

0.14.40.2.4 GitRevisionSHA1

`const std::string smtrat::compile_information::GitRevisionSHA1 = "1bc26d0e6dfae377e44a45002efe7a18ef99e402"`

0.14.40.2.5 GitVersion

`const std::string smtrat::compile_information::GitVersion = "22.12"`

0.14.40.2.6 PackageName

`const std::string smtrat::compile_information::PackageName = "smtrat"`

0.14.40.2.7 ProjectName

`const std::string smtrat::compile_information::ProjectName = "SMT-RAT"`

0.14.40.2.8 SystemName

`const std::string smtrat::compile_information::SystemName = "Linux"`

0.14.40.2.9 SystemVersion

`const std::string smtrat::compile_information::SystemVersion = "6.↩ 0.0-0.deb11.6-amd64"`

0.14.40.2.10 Version

`const std::string smtrat::compile_information::Version = "22.12"`

0.14.40.2.11 Website

`const std::string smtrat::compile_information::Website = "https://ths-rwth.↩ github.io/smtrat/"`

0.14.41 smtrat::covering_ng Namespace Reference

Namespaces

- formula

Data Structures

- struct sampling
- struct sampling< SamplingAlgorithm::LOWER_UPPER_BETWEEN_SAMPLING >
- struct IntervalCompare

  Sorts interval by their lower bounds.
- struct CoveringResult

Typedefs

- template<cadcells::operators::op op>
  using Interval = cadcells::datastructures::SampledDerivationRef< typename cadcells::operators::PropertiesSet< op >::type >
- template<cadcells::operators::op op>
  using IntervalSet = std::set< Interval< op >, IntervalCompare< op > >

Enumerations

- enum class SamplingAlgorithm { LOWER_UPPER_BETWEEN_SAMPLING }

Functions

- carl::Variable first_unassigned_var (const cadcells::Assignment &ass, const cadcells::VariableOrdering &var_order)
- bool is_full_sample (const cadcells::Assignment &ass, const cadcells::VariableOrdering &var_order)
- template<cadcells::operators::op op>
  std::optional< Interval< op > > get_enclosing_interval (cadcells::datastructures::Projections &proj, const boost::container::flat_set< cadcells::Constraint > &implicant, formula::Valuation root_valuation, const cadcells::Assignment &ass)
- template<typename FE , cadcells::operators::op op>
  std::vector< Interval< op > > get_enclosing_intervals (cadcells::datastructures::Projections &proj, const FE &f, const cadcells::Assignment &ass)
- template<cadcells::operators::op op, cadcells::representation::CoveringHeuristic covering_heuristic>
  std::optional< Interval< op > > characterize_covering (const IntervalSet< op > &intervals)
- template<typename FE , cadcells::operators::op op, cadcells::representation::CoveringHeuristic covering_heuristic, smtrat::covering↩_ng::SamplingAlgorithm sampling_algorithm>
  CoveringResult< op > exists (cadcells::datastructures::Projections &proj, FE &f, cadcells::Assignment ass)

0.14.41.1 Typedef Documentation

0.14.41.1.1 Interval

```
template<cadcells::operators::op op>
using smtrat::covering_ng::Interval = typedef cadcells::datastructures::SampledDerivationRef<typename
cadcells::operators::PropertiesSet<op>::type>
```

0.14.41.1.2 IntervalSet

```
template<cadcells::operators::op op>
using smtrat::covering_ng::IntervalSet = typedef std::set<Interval<op>, IntervalCompare<op>
>
```

0.14.41.2 Enumeration Type Documentation

0.14.41.2.1 SamplingAlgorithm

```
enum smtrat::covering_ng::SamplingAlgorithm [strong]
```

Enumerator

LOWER_UPPER_BETWEEN_SAMPLING	First checks lower bound and then upper bound, then checks between the cells if they are covered.

0.14.41.3 Function Documentation

0.14.41.3.1 characterize_covering()

```
template<cadcells::operators::op op, cadcells::representation↩
::CoveringHeuristic covering_heuristic>
std::optional<Interval<op> > smtrat::covering_ng::characterize_covering (
            const IntervalSet< op > & intervals ) [inline]
```

0.14.41.3.2 exists()

```
template<typename FE , cadcells::operators::op op, cadcells::representation↩
::CoveringHeuristic covering_heuristic, smtrat::covering_ng::SamplingAlgorithm sampling_↩
algorithm>
CoveringResult<op> smtrat::covering_ng::exists (
            cadcells::datastructures::Projections & proj,
            FE & f,
            cadcells::Assignment ass )  [inline]
```

0.14.41.3.3 first_unassigned_var()

```
carl::Variable smtrat::covering_ng::first_unassigned_var (
            const cadcells::Assignment & ass,
            const cadcells::VariableOrdering & var_order )  [inline]
```

0.14.41.3.4 get_enclosing_interval()

```
template<cadcells::operators::op op>
std::optional<Interval<op> > smtrat::covering_ng::get_enclosing_interval (
            cadcells::datastructures::Projections & proj,
            const boost::container::flat_set< cadcells::Constraint > & implicant,
            formula::Valuation root_valuation,
            const cadcells::Assignment & ass )  [inline]
```

0.14.41.3.5 get_enclosing_intervals()

```
template<typename FE , cadcells::operators::op op>
std::vector<Interval<op> > smtrat::covering_ng::get_enclosing_intervals (
            cadcells::datastructures::Projections & proj,
            const FE & f,
            const cadcells::Assignment & ass )  [inline]
```

0.14.41.3.6 is_full_sample()

```
bool smtrat::covering_ng::is_full_sample (
            const cadcells::Assignment & ass,
            const cadcells::VariableOrdering & var_order )  [inline]
```

0.14.42 smtrat::covering_ng::formula Namespace Reference

Namespaces

- exhaustive_implicants_helper
- minimal_helper

  Minimal.
- node_ds

Data Structures

- class Minimal
- class ExhaustiveImplicants

Enumerations

- enum class Valuation { TRUE , FALSE , MULTIVARIATE , UNKNOWN }

Functions

- template<class... Ts>
  overloaded (Ts...) -> overloaded< Ts... >
- std::ostream & operator<< (std::ostream &o, Valuation v)

0.14.42.1 Enumeration Type Documentation

0.14.42.1.1 Valuation `enum smtrat::covering_ng::formula::Valuation [strong]`

Enumerator

TRUE	
FALSE	
MULTIVARIATE	
UNKNOWN	

0.14.42.2 Function Documentation

0.14.42.2.1 operator<<() `std::ostream& smtrat::covering_ng::formula::operator<< ( std::ostream & o, Valuation v ) [inline]`

0.14.42.2.2 overloaded() `template<class... Ts> smtrat::covering_ng::formula::overloaded ( Ts... ) -> overloaded< Ts... >`

0.14.43 smtrat::covering_ng::formula::exhaustive_implicants_helper Namespace Reference

Functions

- std::vector< boost::container::flat_set< cadcells::Constraint > > compute_implicants (const node_ds::Node &f, std::size_t pruning, ExhaustiveImplicants::ImplicantComplexityOrdering implicant_complexity_ordering)

0.14.43.1 Function Documentation

0.14.43.1.1 compute_implicants() `std::vector<boost::container::flat_set<cadcells::Constraint> > smtrat::covering_ng::formula::exhaustive_implicants_helper::compute_implicants ( const node_ds::Node & f, std::size_t pruning, ExhaustiveImplicants::ImplicantComplexityOrdering implicant_complexity_ordering )`

0.14.44 smtrat::covering_ng::formula::minimal_helper Namespace Reference

Minimal.

Functions

- void sort_by_complexity (node_ds::Node &f, const std::function< bool(const node_ds::Node &, const node_ds::Node &)> &compare)
- void compute_implicant (const node_ds::Node &f, boost::container::flat_set< cadcells::Constraint > &implicant)

0.14.44.1 Detailed Description

Minimal.

0.14.44.2 Function Documentation

0.14.44.2.1 compute_implicant()

```
void smtrat::covering_ng::formula::minimal_helper::compute_implicant (
    const node_ds::Node & f,
    boost::container::flat_set< cadcells::Constraint > & implicant )
```

0.14.44.2.2 sort_by_complexity()

```
void smtrat::covering_ng::formula::minimal_helper::sort_by_complexity (
    node_ds::Node & f,
    const std::function< bool(const node_ds::Node &, const node_ds::Node &)> & compare )
```

0.14.45 smtrat::covering_ng::formula::node_ds Namespace Reference

Namespaces

- complexity

Data Structures

- class Node
- struct TRUE
- struct FALSE
- struct NOT
- struct AND
- struct OR
- struct IFF
- struct XOR
- struct BOOL
- struct CONSTRAINT
- struct NodeContent

Functions

- void extend_valuation (Node &f, const cadcells::Assignment &ass, bool evaluate_all)

  Updates the valuations.
- void revert_valuation (Node &f, std::size_t level)
- Node to_node (typename cadcells::Polynomial::ContextType c, const FormulaT &f, std::map< std::size_t, Node > &cache)
- Node to_node (typename cadcells::Polynomial::ContextType c, const FormulaT &f)
- FormulaT to_formula (const Node &f)
- auto debug_valuation (const Node &f, const cadcells::Assignment &ass)

0.14.45.1 Function Documentation

0.14.45.1.1 debug_valuation()

```
auto smtrat::covering_ng::formula::node_ds::debug_valuation (
    const Node & f,
    const cadcells::Assignment & ass )
```

0.14.45.1.2 extend_valuation()

```
void smtrat::covering_ng::formula::node_ds::extend_valuation (
    Node & f,
    const cadcells::Assignment & ass,
    bool evaluate_all )
```

Updates the valuations.
Assumes that ass is an extension of the previous assignment the formula has been evaluated with.

Parameters

f	
ass	

0.14.45.1.3 revert_valuation()

```
void smtrat::covering_ng::formula::node_ds::revert_valuation (
    Node & f,
    std::size_t level )
```

0.14.45.1.4 to_formula()

```
FormulaT smtrat::covering_ng::formula::node_ds::to_formula (
    const Node & f )
```

0.14.45.1.5 to_node() [1/2]

```
Node smtrat::covering_ng::formula::node_ds::to_node (
    typename cadcells::Polynomial::ContextType c,
    const FormulaT & f )
```

0.14.45.1.6 to_node() [2/2]

```
Node smtrat::covering_ng::formula::node_ds::to_node (
    typename cadcells::Polynomial::ContextType c,
    const FormulaT & f,
    std::map< std::size_t, Node > & cache )
```

0.14.46 smtrat::covering_ng::formula::node_ds::complexity Namespace Reference

Functions

- bool min_tdeg_ordering (const Node &a, const Node &b)
- bool min_lvl_min_tdeg_ordering (const Node &a, const Node &b)
- bool min_tdeg_min_size_implicant (const boost::container::flat_set< cadcells::Constraint > &a, const boost::container::flat_set< cadcells::Constraint > &b)
- bool min_size_min_tdeg_implicant (const boost::container::flat_set< cadcells::Constraint > &a, const boost::container::flat_set< cadcells::Constraint > &b)

0.14.46.1 Function Documentation

0.14.46.1.1 min_lvl_min_tdeg_ordering()

```
bool smtrat::covering_ng::formula::node_ds::complexity↩
::min_lvl_min_tdeg_ordering (
    const Node & a,
    const Node & b )  [inline]
```

0.14.46.1.2 min_size_min_tdeg_implicant()

```
bool smtrat::covering_ng::formula::node_ds::complexity↩
::min_size_min_tdeg_implicant (
            const boost::container::flat_set< cadcells::Constraint > & a,
            const boost::container::flat_set< cadcells::Constraint > & b )  [inline]
```

0.14.46.1.3 min_tdeg_min_size_implicant()

```
bool smtrat::covering_ng::formula::node_ds::complexity↩
::min_tdeg_min_size_implicant (
            const boost::container::flat_set< cadcells::Constraint > & a,
            const boost::container::flat_set< cadcells::Constraint > & b )  [inline]
```

0.14.46.1.4 min_tdeg_ordering()

```
bool smtrat::covering_ng::formula::node_ds::complexity::min_↩
tdeg_ordering (
            const Node & a,
            const Node & b )  [inline]
```

0.14.47 smtrat::execution Namespace Reference

Data Structures

- struct Assertion
- struct SoftAssertion
- struct Objective
- class ExecutionState

Enumerations

- enum Mode { START =0 , ASSERT =1 , SAT =2 , UNSAT =3 }

0.14.47.1 Enumeration Type Documentation

0.14.47.1.1 Mode

```
enum smtrat::execution::Mode
```

Enumerator

START	
ASSERT	
SAT	
UNSAT	

0.14.48 smtrat::expression Namespace Reference

Namespaces

- simplifier

Data Structures

- class Expression
- struct ITEExpression
- struct QuantifierExpression
- struct UnaryExpression
- struct BinaryExpression
- struct NaryExpression

- struct ExpressionContent
- struct ExpressionTypeChecker
- struct ExpressionConverter
- class ExpressionPool
- class ExpressionVisitor
- class ExpressionModifier

Typedefs

- typedef std::vector< Expression > Expressions

Enumerations

- enum ITEType { ITE }
- enum QuantifierType { EXISTS , FORALL }
- enum UnaryType { NOT }
- enum BinaryType
- enum NaryType { AND , OR , XOR , IFF }

Functions

- std::ostream & operator<< (std::ostream &os, const Expression &expr)
- std::ostream & operator<< (std::ostream &os, const ITEExpression &expr)
- std::ostream & operator<< (std::ostream &os, const QuantifierExpression &expr)
- std::ostream & operator<< (std::ostream &os, const UnaryExpression &expr)
- std::ostream & operator<< (std::ostream &os, const BinaryExpression &expr)
- std::ostream & operator<< (std::ostream &os, const NaryExpression &expr)
- std::ostream & operator<< (std::ostream &os, const ExpressionContent &expr)
- std::ostream & operator<< (std::ostream &os, const ExpressionContent *expr)
- std::ostream & operator<< (std::ostream &os, ITEType)
- std::ostream & operator<< (std::ostream &os, QuantifierType type)
- std::ostream & operator<< (std::ostream &os, UnaryType type)
- std::ostream & operator<< (std::ostream &os, BinaryType type)
- std::ostream & operator<< (std::ostream &os, NaryType type)

0.14.48.1 Typedef Documentation

0.14.48.1.1 Expressions `typedef std::vector<Expression> smtrat::expression::Expressions`

0.14.48.2 Enumeration Type Documentation

0.14.48.2.1 BinaryType `enum smtrat::expression::BinaryType`

0.14.48.2.2 ITEType `enum smtrat::expression::ITEType`

Enumerator

ITE	

0.14.48.2.3 NaryType `enum smtrat::expression::NaryType`

Enumerator

AND	
OR	
XOR	
IFF	

0.14.48.2.4 QuantifierType `enum smtrat::expression::QuantifierType`

Enumerator

EXISTS	
FORALL	

0.14.48.2.5 UnaryType `enum smtrat::expression::UnaryType`

Enumerator

NOT	

0.14.48.3 Function Documentation

0.14.48.3.1 operator<<() [1/13]

```
std::ostream& smtrat::expression::operator<< (
    std::ostream & os,
    BinaryType type ) [inline]
```

0.14.48.3.2 operator<<() [2/13]

```
std::ostream& smtrat::expression::operator<< (
    std::ostream & os,
    const BinaryExpression & expr ) [inline]
```

0.14.48.3.3 operator<<() [3/13]

```
std::ostream & smtrat::expression::operator<< (
    std::ostream & os,
    const Expression & expr )
```

0.14.48.3.4 operator<<() [4/13]

```
std::ostream& smtrat::expression::operator<< (
    std::ostream & os,
    const ExpressionContent & expr ) [inline]
```

0.14.48.3.5 operator<<() [5/13]

```
std::ostream& smtrat::expression::operator<< (
    std::ostream & os,
    const ExpressionContent * expr ) [inline]
```

0.14.48.3.6 operator<<() [6/13]

```
std::ostream& smtrat::expression::operator<< (
    std::ostream & os,
    const ITEExpression & expr ) [inline]
```

0.14.48.3.7 operator<<() [7/13]

```
std::ostream& smtrat::expression::operator<< (
    std::ostream & os,
    const NaryExpression & expr ) [inline]
```

0.14.48.3.8 operator<<() [8/13]

```
std::ostream& smtrat::expression::operator<< (
    std::ostream & os,
    const QuantifierExpression & expr ) [inline]
```

0.14.48.3.9 operator<<() [9/13]

```
std::ostream& smtrat::expression::operator<< (
    std::ostream & os,
    const UnaryExpression & expr ) [inline]
```

0.14.48.3.10 operator<<() [10/13]

```
std::ostream& smtrat::expression::operator<< (
    std::ostream & os,
    ITEType ) [inline]
```

0.14.48.3.11 operator<<() [11/13]

```
std::ostream& smtrat::expression::operator<< (
    std::ostream & os,
    NaryType type ) [inline]
```

0.14.48.3.12 operator<<() [12/13]

```
std::ostream& smtrat::expression::operator<< (
    std::ostream & os,
    QuantifierType type ) [inline]
```

0.14.48.3.13 operator<<() [13/13]

```
std::ostream& smtrat::expression::operator<< (
    std::ostream & os,
    UnaryType type ) [inline]
```

0.14.49 smtrat::expression::simplifier Namespace Reference

Data Structures

- struct BaseSimplifier
- struct DuplicateSimplifier
- struct MergeSimplifier
- struct NegationSimplifier
- struct SimplifierChainCaller
- struct SimplifierChainCaller< 0 >
- class Simplifier
- struct SingletonSimplifier

Typedefs

- typedef std::tuple< MergeSimplifier, DuplicateSimplifier, SingletonSimplifier > SimplifierChain

0.14.49.1 Typedef Documentation

0.14.49.1.1 SimplifierChain `typedef std::tuple< MergeSimplifier, DuplicateSimplifier, SingletonSimplifier > smtrat::expression::simplifier::SimplifierChain`

0.14.50 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)

0.14.50.1 Typedef Documentation

0.14.50.1.1 RewriteRules `typedef std::map<carl::Variable, std::pair<TermT, carl::BitVector> > smtrat::groebner::RewriteRules`

0.14.50.2 Function Documentation

0.14.50.2.1 rewritePolynomial()

```
template<typename Polynomial >
static Polynomial smtrat::groebner::rewritePolynomial (
            const Polynomial & inputPolynomial,
            const RewriteRules & rules )  [static]
```

0.14.51 smtrat::helper Namespace Reference

Functions

- template<typename Pol >
  static size_t level_of (const VariableOrdering &order, const Pol &poly)

0.14.51.1 Function Documentation

0.14.51.1.1 level_of()

```
template<typename Pol >
static size_t smtrat::helper::level_of (
            const VariableOrdering & order,
            const Pol & poly )  [static]
```

0.14.52 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 class Updated { LEFT , RIGHT , BOTH , NONE }

Functions

- std::vector< Poly > getNonlinearMonomials (const Poly &_expr)

  Obtains the non-linear monomials of the given polynomial.
- std::pair< ConstraintT, ConstraintT > intervalToConstraint (const Poly &_lhs, const smtrat::DoubleInterval _interval)

  Creates a new constraint from an existing interval.
- bool intervalBoxContainsEmptyInterval (const EvalDoubleIntervalMap &_intervals)

  Checks mIntervals if it contains an empty interval.
- const LRAModule< LRASettings1 >::LRAVariable * getOriginalLraVar (carl::Variable::Arg _var, const LRAModule< LRASettings1 > &_lra)

0.14.52.1 Typedef Documentation

0.14.52.1.1 Contractor

`template<template< typename > class Operator>`
`using smtrat::icp::Contractor = typedef carl::Contraction<Operator, Poly>`

0.14.52.1.2 LRAVariable

`typedef LRAModule<LRASettings1>::LRAVariable smtrat::icp::LRAVariable`

0.14.52.1.3 set_icpVariable

`typedef std::set<const IcpVariable*, icpVariableComp> smtrat::icp::set_icpVariable`

0.14.52.2 Enumeration Type Documentation

0.14.52.2.1 Updated

`enum smtrat::icp::Updated [strong]`

Enumerator

LEFT	
RIGHT	
BOTH	
NONE	

0.14.52.3 Function Documentation

0.14.52.3.1 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.

0.14.52.3.2 getOriginalLraVar()

```
const LRAModule< LRASettings1 >::LRAVariable * smtrat::icp↩
::getOriginalLraVar (
            carl::Variable::Arg _var,
            const LRAModule< LRASettings1 > & _lra )
```

0.14.52.3.3 intervalBoxContainsEmptyInterval()

```
bool smtrat::icp::intervalBoxContainsEmptyInterval
(
            const EvalDoubleIntervalMap & _intervals )
```

Checks mIntervals if it contains an empty interval.

Returns

0.14.52.3.4 intervalToConstraint()

```
std::pair< ConstraintT, ConstraintT > smtrat::icp::interval↩
ToConstraint (
            const Poly & _lhs,
            const smtrat::DoubleInterval _interval )
```

Creates a new constraint from an existing interval.

Parameters

_interval	

Returns

pair <lowerBoundConstraint∗, upperBoundConstraint∗>

0.14.53 smtrat::lra Namespace Reference

Data Structures

- class Variable
- class Bound

  Stores a bound, which could be an upper "<= b" or a lower bound ">= b" for a bound value b.
- class Numeric
- class TableauEntry
- class Tableau
- class Value

Typedefs

- typedef size_t EntryID

Functions

- Numeric abs (const Numeric &_value)

  Calculates the absolute value of the given Numeric.
- Numeric mod (const Numeric &_valueA, const Numeric &_valueB)

  Calculates the result of the first argument modulo the second argument.
- Numeric lcm (const Numeric &_valueA, const Numeric &_valueB)

  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

0.14.53.1 Typedef Documentation

0.14.53.1.1 EntryID `typedef size_t smtrat::lra::EntryID`

0.14.53.2 Function Documentation

0.14.53.2.1 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.

0.14.53.2.2 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.

0.14.53.2.3 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.

0.14.53.2.4 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.

0.14.53.2.5 operator∗()

```
Numeric smtrat::lra::operator* (
    const Numeric & _valueA,
    const Numeric & _valueB )
```

Calculates the product of the two given Numerics.

Parameters

_valueA	The first factor.
_valueB	The second factor.

Returns

The product of the two given Numerics.

0.14.53.2.6 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.

0.14.53.2.7 operator+()

```
Numeric smtrat::lra::operator+ (
    const Numeric & _valueA,
    const Numeric & _valueB )
```

Calculates the sum of the two given Numerics.

Parameters

_valueA	The first summand.
_valueB	The second summand.

Returns

The sum of the two given Numerics.

0.14.53.2.8 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.

0.14.53.2.9 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.

0.14.53.2.10 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.

0.14.53.2.11 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.

0.14.53.2.12 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.

0.14.53.2.13 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.

0.14.53.2.14 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.

0.14.53.2.15 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.

0.14.53.2.16 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.

0.14.53.2.17 operator<<() [2/2]

```
template<typename T1 >
std::ostream& smtrat::lra::operator<< (
    std::ostream & _out,
    const Value< T1 > & _value )
```

0.14.53.3 Variable Documentation

0.14.53.3.1 LAST_ENTRY_ID

```
EntryID smtrat::lra::LAST_ENTRY_ID = 0 [static]
```

0.14.54 smtrat::lve Namespace Reference

Functions

- Rational evaluate (carl::Variable v, const Poly &p, const Rational &r)
- carl::Sign sgn (carl::Variable v, const Poly &p, const RAN &r)
- std::optional< ModelValue > get_root (carl::Variable v, const Poly &p)
- ModelValue get_non_root (carl::Variable v, const Poly &p)
- std::optional< ModelValue > get_value_for_sgn (carl::Variable v, const Poly &p, carl::Sign sign)
- carl::Sign sgn_of_invariant_poly (carl::Variable v, const Poly &p)

0.14.54.1 Function Documentation

0.14.54.1.1 evaluate()

```
Rational smtrat::lve::evaluate (
    carl::Variable v,
    const Poly & p,
    const Rational & r )
```

0.14.54.1.2 get_non_root()

```
ModelValue smtrat::lve::get_non_root (
    carl::Variable v,
    const Poly & p )
```

0.14.54.1.3 get_root()

```
std::optional<ModelValue> smtrat::lve::get_root (
    carl::Variable v,
    const Poly & p )
```

0.14.54.1.4 get_value_for_sgn()

```
std::optional<ModelValue> smtrat::lve::get_value_for_sgn (
    carl::Variable v,
    const Poly & p,
    carl::Sign sign )
```

0.14.54.1.5 sgn()

```
carl::Sign smtrat::lve::sgn (
    carl::Variable v,
    const Poly & p,
    const RAN & r )
```

0.14.54.1.6 sgn_of_invariant_poly()

```
carl::Sign smtrat::lve::sgn_of_invariant_poly (
    carl::Variable v,
    const Poly & p )
```

0.14.55 smtrat::maxsmt Namespace Reference

Contains strategy implementations for max SMT computations.

Data Structures

- class MaxSMTBackend
- class MaxSMTBackend< Solver, MaxSMTStrategy::FU_MALIK_INCREMENTAL >
- class MaxSMTBackend< Solver, MaxSMTStrategy::LINEAR_SEARCH >
- class MaxSMTBackend< Solver, MaxSMTStrategy::MSU3 >

0.14.55.1 Detailed Description

Contains strategy implementations for max SMT computations.

0.14.56 smtrat::mcsat Namespace Reference

Namespaces

- arithmetic
- constraint_type
- fm

  A Fourier-Motzkin based backend.
- icp
- nlsat
- onecell
- onecellcad
- smtaf
- variableordering
- vs

Data Structures

- struct SequentialAssignment
- struct FastParallelExplanation

  This explanation executes all given explanation in parallel processes and waits for the fastest explanation, returning the fastest delivered explanation, terminating all other parallel processes.
- struct FullParallelExplanation

  This explanation executes all given explanation parallel in multiple threads and waits until every single one has finished, returning the result of the first listed explanation.
- struct ParallelExplanation
- struct SequentialExplanation
- class Bookkeeping

  Represent the trail, i.e.
- class ClauseChain

  An explanation is either a single clause or a chain of clauses, satisfying the following properties:
- class MCSATBackend
- struct InformationGetter
- struct TheoryLevel
- class MCSATMixin

Typedefs

- typedef allocator< std::optional< Explanation >, managed_shared_memory::segment_manager > ShmemAllocator
- typedef vector< std::optional< Explanation >, ShmemAllocator > SharedVector
- using ModelValues = std::vector< std::pair< carl::Variable, ModelValue > >
- using AssignmentOrConflict = std::variant< ModelValues, FormulasT >
- using Explanation = std::variant< FormulaT, ClauseChain >

Enumerations

- enum class ConstraintType { Constant , Assigned , Univariate , Unassigned }

  This type categorizes constraints with respect to a given (partial) model and the next variable to be assigned.
- enum class VariableOrdering { GreedyMaxUnivariate , FeatureBased , FeatureBasedZ3 , FeatureBasedBrown }

Functions

- FormulaT resolveExplanation (const Explanation &expl)
- std::ostream & operator<< (std::ostream &os, const Bookkeeping &bk)
- FormulaT _transformToImplicationChain (const FormulaT &formula, const Model &model, ClauseChain &chain, bool outermost, bool withEquivalences)
- std::ostream & operator<< (std::ostream &stream, const ClauseChain::Link &link)
- std::ostream & operator<< (std::ostream &stream, const ClauseChain &chain)
- std::ostream & operator<< (std::ostream &os, ConstraintType ct)
- std::string get_name (VariableOrdering ordering)
- std::ostream & operator<< (std::ostream &os, VariableOrdering ordering)
- template<VariableOrdering vot, typename Constraints >
  std::vector< carl::Variable > calculate_variable_order (const Constraints &c)
- template<VariableOrdering vot>
  std::vector< carl::Variable > calculate_variable_order (const std::vector< ConstraintT > &constraints)
- template<typename Settings >
  std::ostream & operator<< (std::ostream &os, const MCSATBackend< Settings > &backend)

0.14.56.1 Typedef Documentation

0.14.56.1.1 AssignmentOrConflict

`using smtrat::mcsat::AssignmentOrConflict = typedef std::variant<ModelValues,FormulasT>`

0.14.56.1.2 Explanation

`using smtrat::mcsat::Explanation = typedef std::variant<FormulaT, ClauseChain>`

0.14.56.1.3 ModelValues

`using smtrat::mcsat::ModelValues = typedef std::vector<std::pair<carl::Variable, ModelValue> >`

0.14.56.1.4 SharedVector

`typedef vector<std::optional<Explanation> , ShmemAllocator> smtrat::mcsat::SharedVe`

0.14.56.1.5 ShmemAllocator

`typedef allocator<std::optional<Explanation> , managed_shared_memory::segment_manager> smtrat::mcsat::ShmemAllocator`

0.14.56.2 Enumeration Type Documentation

0.14.56.2.1 ConstraintType

`enum smtrat::mcsat::ConstraintType [strong]`

This type categorizes constraints with respect to a given (partial) model and the next variable to be assigned. Note that `Constant` implies `Assigned`.

Enumerator

Constant	
Assigned	The constraint contains no variables.
Univariate	The constraint is fully assigned.
Unassigned	The constraint has a single unassigned variable being the next one. The constraint has an unassigned variable that is not the next one.

0.14.56.2.2 VariableOrdering

`enum smtrat::mcsat::VariableOrdering [strong]`

Enumerator

GreedyMaxUnivariate	
FeatureBased	
FeatureBasedZ3	
FeatureBasedBrown	

0.14.56.3 Function Documentation

0.14.56.3.1 _transformToImplicationChain()

```
FormulaT smtrat::mcsat::_transformToImplicationChain
(
    const FormulaT & formula,
    const Model & model,
    ClauseChain & chain,
```

```
            bool outermost,
            bool withEquivalences )
```

0.14.56.3.2 calculate_variable_order() [1/2]

```
template<VariableOrdering vot, typename Constraints
>
std::vector<carl::Variable> smtrat::mcsat::calculate_variable_order (
            const Constraints & c )
```

0.14.56.3.3 calculate_variable_order() [2/2]

```
template<VariableOrdering vot>
std::vector<carl::Variable> smtrat::mcsat::calculate_variable_order (
            const std::vector< ConstraintT > & constraints )
```

0.14.56.3.4 get_name()

```
std::string smtrat::mcsat::get_name (
            VariableOrdering ordering ) [inline]
```

0.14.56.3.5 operator<<() [1/6]

```
std::ostream& smtrat::mcsat::operator<< (
            std::ostream & os,
            const Bookkeeping & bk ) [inline]
```

0.14.56.3.6 operator<<() [2/6]

```
template<typename Settings >
std::ostream& smtrat::mcsat::operator<< (
            std::ostream & os,
            const MCSATBackend< Settings > & backend )
```

0.14.56.3.7 operator<<() [3/6]

```
std::ostream& smtrat::mcsat::operator<< (
            std::ostream & os,
            ConstraintType ct ) [inline]
```

0.14.56.3.8 operator<<() [4/6]

```
std::ostream& smtrat::mcsat::operator<< (
            std::ostream & os,
            VariableOrdering ordering ) [inline]
```

0.14.56.3.9 operator<<() [5/6]

```
std::ostream& smtrat::mcsat::operator<< (
            std::ostream & stream,
            const ClauseChain & chain ) [inline]
```

0.14.56.3.10 operator<<() [6/6]

```
std::ostream& smtrat::mcsat::operator<< (
            std::ostream & stream,
            const ClauseChain::Link & link ) [inline]
```

0.14.56.3.11 resolveExplanation()

```
FormulaT smtrat::mcsat::resolveExplanation (
            const Explanation & expl ) [inline]
```

0.14.57 smtrat::mcsat::arithmetic Namespace Reference

Data Structures

- struct AssignmentFinder
- class AssignmentFinder_detail
- class AssignmentFinder_ctx
- class Covering

  Semantics: The space is divided into a number of intervals: (-oo,a)[a,a](a,b)[b,b](b,oo) A bit is set if the constraints refutes the corresponding interval.
- class RootIndexer

Functions

- std::ostream & operator<< (std::ostream &os, const Covering &ri)
- template<typename RANT >
  std::ostream & operator<< (std::ostream &os, const RootIndexer< RANT > &ri)

0.14.57.1 Function Documentation

0.14.57.1.1 operator<<() [1/2]

```
std::ostream& smtrat::mcsat::arithmetic::operator<< (
    std::ostream & os,
    const Covering & ri ) [inline]
```

0.14.57.1.2 operator<<() [2/2]

```
template<typename RANT >
std::ostream& smtrat::mcsat::arithmetic::operator<< (
    std::ostream & os,
    const RootIndexer< RANT > & ri ) [inline]
```

0.14.58 smtrat::mcsat::constraint_type Namespace Reference

Functions

- template<typename T >
  ConstraintType categorize (const T &t, const Model &model, carl::Variable next)
- template<typename T >
  bool is_constant (const T &t)

  Checks whether the constraint is constant, i.e.
- template<typename T >
  bool isAssigned (const T &t, const Model &model)

  Checks whether all variables contained in the constraint are assigned in the model.
- template<typename T >
  bool is_univariate (const T &t, const Model &model, carl::Variable next)

  Checks whether the constraint contains only a single unassigned variable, and this is the next one.
- template<typename T >
  bool isUnassigned (const T &t, const Model &model, carl::Variable next)

  Checks whether the constraint contains an unassigned variable that is not the next one.

0.14.58.1 Function Documentation

0.14.58.1.1 categorize() template<typename T >

```
ConstraintType smtrat::mcsat::constraint_type::categorize (
            const T & t,
            const Model & model,
            carl::Variable next )
```

0.14.58.1.2 is_constant() template<typename T >

```
bool smtrat::mcsat::constraint_type::is_constant (
            const T & t )
```

Checks whether the constraint is constant, i.e.
whether it contains no variables.

0.14.58.1.3 is_univariate() template<typename T >

```
bool smtrat::mcsat::constraint_type::is_univariate (
            const T & t,
            const Model & model,
            carl::Variable next )
```

Checks whether the constraint contains only a single unassigned variable, and this is the next one.

0.14.58.1.4 isAssigned() template<typename T >

```
bool smtrat::mcsat::constraint_type::isAssigned (
            const T & t,
            const Model & model )
```

Checks whether all variables contained in the constraint are assigned in the model.
In particular, a `Constant` constraint is also `Assigned`.

0.14.58.1.5 isUnassigned() template<typename T >

```
bool smtrat::mcsat::constraint_type::isUnassigned (
            const T & t,
            const Model & model,
            carl::Variable next )
```

Checks whether the constraint contains an unassigned variable that is not the next one.

0.14.59 smtrat::mcsat::fm Namespace Reference

A Fourier-Motzkin based backend.

Data Structures

- struct Bound
- struct ConflictGenerator
- struct DefaultComparator

  Does not order anything.
- struct MaxSizeComparator

  This heuristic chooses the explanation excluding the largest interval.
- struct MinSizeComparator

  This heuristic chooses the explanation excluding the smallest interval.
- struct MinVarCountComparator

  This heuristic tries to minimize the number of variables occuring in the explanation.
- struct DefaultSettings
- struct IgnoreCoreSettings
- struct Explanation

Functions

- bool isSubset (const carl::Variables &subset, const carl::Variables &superset)
- std::ostream & operator<< (std::ostream &os, const Bound &b)

0.14.59.1 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.

0.14.59.2 Function Documentation

0.14.59.2.1 isSubset()

```
bool smtrat::mcsat::fm::isSubset (
    const carl::Variables & subset,
    const carl::Variables & superset ) [inline]
```

0.14.59.2.2 operator<<()

```
std::ostream& smtrat::mcsat::fm::operator<< (
    std::ostream & os,
    const Bound & b ) [inline]
```

0.14.60 smtrat::mcsat::icp Namespace Reference

Data Structures

- class Dependencies
- struct Explanation
- struct QueueEntry
- class IntervalPropagation

0.14.61 smtrat::mcsat::nlsat Namespace Reference

Namespaces

- helper

Data Structures

- struct Explanation
- class ExplanationGenerator

0.14.62 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.

0.14.62.1 Function Documentation

0.14.62.1.1 buildAbove()

```
template<typename MVRootParams >
FormulaT smtrat::mcsat::nlsat::helper::buildAbove (
            carl::Variable var,
            const MVRootParams & mvp )
```

Construct an atomic formula representing a variable being greater than the given multivariate root.
"v > root(..)"

0.14.62.1.2 buildBelow()

```
template<typename MVRootParams >
FormulaT smtrat::mcsat::nlsat::helper::buildBelow (
            carl::Variable var,
            const MVRootParams & mvp )
```

Construct an atomic formula representing a variable being less than the given multivariate root.
"v < root(..)"

0.14.62.1.3 buildEquality()

```
template<typename MVRootParams >
FormulaT smtrat::mcsat::nlsat::helper::buildEquality (
            carl::Variable var,
            const MVRootParams & mvp )
```

Construct an atomic formula representing a variable being equal to the given multivariate root.
"v = root(..)"

0.14.62.1.4 buildFormulaFromVC()

```
FormulaT smtrat::mcsat::nlsat::helper::buildFormulaFromVC (
            VariableComparisonT && vc )  [inline]
```

Construct a formula representing a variable comparison.
Simplify to a regular constraint if possible.

0.14.62.1.5 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.

0.14.63 smtrat::mcsat::onecell Namespace Reference

Data Structures

- struct Explanation
- struct Base
- struct LDBSettings
- struct LDBFilteredAllSelectiveSettings
- struct BCSettings
- struct BCFilteredSettings
- struct BCFilteredAllSettings
- struct BCFilteredBoundsSettings
- struct BCFilteredSamplesSettings
- struct BCFilteredAllSelectiveSettings
- struct BCApproximationSettings

Typedefs

- using Settings = BCApproximationSettings

Functions

- template<typename Settings >
  std::optional< cadcells::CNF > onecell (const std::vector< cadcells::Atom > &constraints, const cadcells↩ ::Polynomial::ContextType &context, const cadcells::Assignment &sample)

  An MCSAT-style single cell explanation function.

Variables

- bool clause_chain_with_equivalences = false

0.14.63.1 Typedef Documentation

0.14.63.1.1 Settings `using smtrat::mcsat::onecell::Settings = typedef BCApproximationSettings`

0.14.63.2 Function Documentation

0.14.63.2.1 onecell()

```
template<typename Settings >
std::optional<cadcells::CNF> smtrat::mcsat::onecell::onecell (
            const std::vector< cadcells::Atom > & constraints,
            const cadcells::Polynomial::ContextType & context,
            const cadcells::Assignment & sample )
```

An MCSAT-style single cell explanation function.
A set of constraints is called infeasible w.r.t. an assignment if the defining polynomials are univariate under the sample and there does not 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 set of constraints whose conjunction describes an unsatisfying cell that can be concluded from the input constraints.

0.14.63.3 Variable Documentation

0.14.63.3.1 clause_chain_with_equivalences `bool smtrat::mcsat::onecell::clause_chain_with_equivalences = false`

0.14.64 smtrat::mcsat::onecellcad Namespace Reference

Namespaces

- levelwise
- recursive

  OneCell Explanation.

Data Structures

- class RealAlgebraicPoint

  Represent a multidimensional point whose components are algebraic reals.
- struct TagPoly

  Tagged Polynomials.
- struct Section

  Represent a cell's (closed-interval-boundary) component along th k-th axis.
- struct Sector

  Represent a cell's open-interval boundary object along one single axis by two irreducible, multivariate polynomials of level k.
- class OneCellCAD

Typedefs

- using RANMap = std::map< carl::Variable, RAN >
- using CADCell = std::vector< std::variant< Sector, Section > >

  Represent a single cell [brown15].

Enumerations

- enum class InvarianceType { ORD_INV , SIGN_INV }

  Invariance Types.

Functions

- template<typename PolyType >
  bool hasOnlyNonConstIrreducibles (const std::vector< PolyType > &polys)
- template<typename PolyType >
  bool isNonConstIrreducible (const PolyType &poly)
- template<typename T >
  bool hasUniqElems (const std::vector< T > &container)
- template<typename T >
  bool isSubset (const std::vector< T > &c1, const std::vector< T > &c2)
- template<typename T >
  std::vector< T > asVector (const std::set< T > s)

- template<typename PolyType >
  bool polyVarsAreAllInList (const std::vector< PolyType > &polys, const std::vector< carl::Variable > &variables)
- template<typename Number >
  bool operator== (RealAlgebraicPoint< Number > &lhs, RealAlgebraicPoint< Number > &rhs)

  Check if two RealAlgebraicPoints are equal.
- template<typename Number >
  std::ostream & operator<< (std::ostream &os, const RealAlgebraicPoint< Number > &r)

  Streaming operator for a RealAlgebraicPoint.
- std::ostream & operator<< (std::ostream &os, const InvarianceType &inv)
- bool operator< (const InvarianceType &l, const InvarianceType &r)
- std::ostream & operator<< (std::ostream &os, const TagPoly &p)
- bool operator== (const TagPoly &lhs, const TagPoly &rhs)
- std::ostream & operator<< (std::ostream &os, const std::vector< TagPoly > &polys)
- std::ostream & operator<< (std::ostream &os, const std::vector< std::vector< TagPoly >> &lvls)
- std::size_t getDegree (TagPoly p, carl::Variable v)
- std::optional< std::size_t > levelOf (const std::vector< carl::Variable > &variableOrder, const Poly &poly)

  Find the index of the highest variable (wrt.
- std::vector< TagPoly > nonConstIrreducibleFactors (std::vector< carl::Variable > variableOrder, Poly poly, InvarianceType tag)
- void appendOnCorrectLevel (const Poly &poly, InvarianceType tag, std::vector< std::vector< TagPoly >> &polys, std::vector< carl::Variable > variableOrder)
- std::ostream & operator<< (std::ostream &os, const Section &s)
- std::ostream & operator<< (std::ostream &os, const Sector &s)
- std::ostream & operator<< (std::ostream &os, const CADCell &cell)
- std::vector< Poly > asMultiPolys (const std::vector< TagPoly > polys)
- bool contains (const std::vector< TagPoly > &polys, const Poly &poly)
- template<typename T >
  bool contains (const std::vector< T > &list, const T &elem)
- MultivariateRootT asRootExpr (carl::Variable rootVariable, Poly poly, std::size_t rootIdx)
- RealAlgebraicPoint< 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)

0.14.64.1 Typedef Documentation

0.14.64.1.1 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.

0.14.64.1.2 RANMap

`using smtrat::mcsat::onecellcad::RANMap = typedef std::map<carl::Variable, RAN>`

0.14.64.2 Enumeration Type Documentation

0.14.64.2.1 InvarianceType

`enum smtrat::mcsat::onecellcad::InvarianceType [strong]`

Invariance Types.

Enumerator

ORD_INV	
SIGN_INV	

0.14.64.3 Function Documentation

0.14.64.3.1 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]
```

0.14.64.3.2 appendOnCorrectLevel()

```
void smtrat::mcsat::onecellcad::appendOnCorrectLevel (
    const Poly & poly,
    InvarianceType tag,
    std::vector< std::vector< TagPoly >> & polys,
    std::vector< carl::Variable > variableOrder ) [inline]
```

0.14.64.3.3 asMultiPolys()

```
std::vector<Poly> smtrat::mcsat::onecellcad::asMultiPolys (
    const std::vector< TagPoly > polys ) [inline]
```

0.14.64.3.4 asRANPoint()

```
RealAlgebraicPoint<smtrat::Rational> smtrat::mcsat::onecellcad::asRANPoint (
    const mcsat::Bookkeeping & data ) [inline]
```

0.14.64.3.5 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.

0.14.64.3.6 asVector()

`template<typename T >`

```
std::vector<T> smtrat::mcsat::onecellcad::asVector (
            const std::set< T > s )
```

0.14.64.3.7 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').

0.14.64.3.8 contains() [1/2]

```
template<typename T >
bool smtrat::mcsat::onecellcad::contains (
            const std::vector< T > & list,
            const T & elem )
```

0.14.64.3.9 contains() [2/2]

```
bool smtrat::mcsat::onecellcad::contains (
            const std::vector< TagPoly > & polys,
            const Poly & poly )  [inline]
```

0.14.64.3.10 discriminant()

```
Poly smtrat::mcsat::onecellcad::discriminant (
            const carl::Variable & mainVariable,
            const Poly & p )  [inline]
```

Projection related utilities for onecellcad.

0.14.64.3.11 duplicateElimination() [1/2]

```
void smtrat::mcsat::onecellcad::duplicateElimination (
            std::vector< Poly > & vec )  [inline]
```

0.14.64.3.12 duplicateElimination() [2/2]

```
std::vector<std::pair<Poly, Poly> > smtrat::mcsat↩
::onecellcad::duplicateElimination (
            std::vector< std::pair< Poly, Poly >> vec )  [inline]
```

0.14.64.3.13 fullSpaceCell()

```
CADCell smtrat::mcsat::onecellcad::fullSpaceCell (
            std::size_t cellComponentCount )  [inline]
```

0.14.64.3.14 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

0.14.64.3.15 hasOnlyNonConstIrreducibles()

```
template<typename PolyType >
bool smtrat::mcsat::onecellcad::hasOnlyNonConstIrreducibles (
            const std::vector< PolyType > & polys )
```

0.14.64.3.16 hasUniqElems()

```
template<typename T >
bool smtrat::mcsat::onecellcad::hasUniqElems (
            const std::vector< T > & container )
```

0.14.64.3.17 isNonConstIrreducible()

```
template<typename PolyType >
bool smtrat::mcsat::onecellcad::isNonConstIrreducible (
            const PolyType & poly )
```

0.14.64.3.18 isSubset()

```
template<typename T >
bool smtrat::mcsat::onecellcad::isSubset (
            const std::vector< T > & c1,
            const std::vector< T > & c2 )
```

0.14.64.3.19 leadcoefficient()

```
Poly smtrat::mcsat::onecellcad::leadcoefficient (
            const carl::Variable & mainVariable,
            const Poly & p )  [inline]
```

0.14.64.3.20 levelOf()

```
std::optional<std::size_t> smtrat::mcsat::onecellcad::levelOf (
            const std::vector< carl::Variable > & variableOrder,
            const Poly & poly )  [inline]
```

Find the index of the highest variable (wrt.
the ordering in 'variableOrder') that occurs with positive degree in 'poly'. Although 'level' is a math concept that starts counting at 1 we start counting at 0 and represent "no level/variable" as std::nullopt because it simplifies using the level directly as an index into arrays or vectors. Examples:

- polyLevel(2) == nullopt wrt. any variable order
- polyLevel(0∗x+2) == nullopt wrt. any variable order
- polyLevel(x+2) == 0 wrt. [x < y < z]
- polyLevel(x+2) == 1 wrt. [y < x < z]
- polyLevel(x+2) == 2 wrt. [y < z < x]
- polyLevel(x∗y+2) == 1 wrt. [x < y < z] because of y
- polyLevel(x∗y+2) == 1 wrt. [y < x < z] because of x
- polyLevel(x∗y+2) == 2 wrt. [x < z < y] because of y Preconditions:
- 'variables(poly)' must be a subset of 'variableOrder'.

0.14.64.3.21 nonConstIrreducibleFactors()

```
std::vector<TagPoly> smtrat::mcsat::onecellcad::nonConstIrreducibleFactors (
    std::vector< carl::Variable > variableOrder,
    Poly poly,
    InvarianceType tag ) [inline]
```

0.14.64.3.22 operator<()

```
bool smtrat::mcsat::onecellcad::operator< (
    const InvarianceType & l,
    const InvarianceType & r ) [inline]
```

0.14.64.3.23 operator<<() [1/8]

```
std::ostream& smtrat::mcsat::onecellcad::operator<< (
    std::ostream & os,
    const CADCell & cell ) [inline]
```

0.14.64.3.24 operator<<() [2/8]

```
std::ostream& smtrat::mcsat::onecellcad::operator<< (
    std::ostream & os,
    const InvarianceType & inv ) [inline]
```

0.14.64.3.25 operator<<() [3/8]

```
template<typename Number >
std::ostream& smtrat::mcsat::onecellcad::operator<< (
    std::ostream & os,
    const RealAlgebraicPoint< Number > & r )
```

Streaming operator for a RealAlgebraicPoint.

0.14.64.3.26 operator<<() [4/8]

```
std::ostream& smtrat::mcsat::onecellcad::operator<< (
    std::ostream & os,
    const Section & s ) [inline]
```

0.14.64.3.27 operator<<() [5/8]

```
std::ostream& smtrat::mcsat::onecellcad::operator<< (
    std::ostream & os,
    const Sector & s ) [inline]
```

0.14.64.3.28 operator<<() [6/8]

```
std::ostream& smtrat::mcsat::onecellcad::operator<< (
    std::ostream & os,
    const std::vector< std::vector< TagPoly >> & lvls ) [inline]
```

0.14.64.3.29 operator<<() [7/8]

```
std::ostream& smtrat::mcsat::onecellcad::operator<< (
    std::ostream & os,
    const std::vector< TagPoly > & polys ) [inline]
```

0.14.64.3.30 operator<<() [8/8]

```
std::ostream& smtrat::mcsat::onecellcad::operator<< (
    std::ostream & os,
    const TagPoly & p ) [inline]
```

0.14.64.3.31 operator==() [1/2]

```
bool smtrat::mcsat::onecellcad::operator== (
            const TagPoly & lhs,
            const TagPoly & rhs ) [inline]
```

0.14.64.3.32 operator==() [2/2]

```
template<typename Number >
bool smtrat::mcsat::onecellcad::operator== (
            RealAlgebraicPoint< Number > & lhs,
            RealAlgebraicPoint< Number > & rhs )
```

Check if two RealAlgebraicPoints are equal.

0.14.64.3.33 optimized_singleLevelFullProjection()

```
bool smtrat::mcsat::onecellcad::optimized_↩
singleLevelFullProjection (
            carl::Variable mainVar,
            size_t currentLevel,
            std::vector< std::vector< TagPoly >> & projectionLevels,
            OneCellCAD & cad ) [inline]
```

0.14.64.3.34 polyVarsAreAllInList()

```
template<typename PolyType >
bool smtrat::mcsat::onecellcad::polyVarsAreAllInList (
            const std::vector< PolyType > & polys,
            const std::vector< carl::Variable > & variables )
```

0.14.64.3.35 prefix()

```
template<typename T >
std::vector<T> smtrat::mcsat::onecellcad::prefix (
            const std::vector< T > vars,
            std::size_t prefixSize )
```

0.14.64.3.36 resultant()

```
Poly smtrat::mcsat::onecellcad::resultant (
            const carl::Variable & mainVariable,
            const Poly & p1,
            const Poly & p2 ) [inline]
```

0.14.64.3.37 singleLevelFullProjection()

```
void smtrat::mcsat::onecellcad::singleLevelFullProjection
(
            std::vector< carl::Variable > & variableOrder,
            carl::Variable mainVar,
            size_t currentLevel,
            std::vector< std::vector< TagPoly >> & projectionLevels ) [inline]
```

0.14.65 smtrat::mcsat::onecellcad::levelwise Namespace Reference

Data Structures

- struct SectionHeuristic1
- struct SectionHeuristic2
- struct SectionHeuristic3
- struct SectorHeuristic1
- struct SectorHeuristic2
- struct SectorHeuristic3
- struct Explanation
- class LevelwiseCAD

0.14.66 smtrat::mcsat::onecellcad::recursive Namespace Reference

OneCell Explanation.

Data Structures

- struct CoverNullification
- struct DontCoverNullification
- struct NoHeuristic
- struct DegreeAscending
- struct DegreeDescending
- struct Explanation
- class RecursiveCAD

Enumerations

- enum class ShrinkResult { SUCCESS , FAIL }

Functions

- void setNullification (bool n)
- std::ostream & operator<< (std::ostream &os, const ShrinkResult &s)

Variables

- bool cover_nullification

0.14.66.1 Detailed Description

OneCell Explanation.

0.14.66.1.1 Rationale To better understand the CAD implementation this document concisely explains important CAD terminology from the literature and high-level implementation design decisions. Low-level design decisions for structs and classes are found in the implementation files themselves.

0.14.66.1.2 Universe The n-dimensional space in which a CADCell exists. The universe has n axes (=std basis vectors), numbered 0 to n-1, and each axis is associated with one variable.

0.14.66.1.3 Variables order A variable order like x,y,z tell us that the universe is 3-dimensional, gives a name to the each coordinate axis (thus also defines an axis ordering), tells us how to interpret a point like (5,2,1), and allows us to define properties like "level" for points and polynomials and "cylindrical" for CADCells.

0.14.66.1.4 Level of a point Number of components that a point has but interpreted with respect to the first variables in a variable ordering. E.g., if the universe has 3 axes/variables like x,y,z in that (increasing) order, then a point of level 2 has exactly 2 components representing the coordinates for the first two variables, x and y.

0.14.66.1.5 Level of a polynomial The number/index of the highest variable, with respect to a variable ordering, that appears with a positive degree in a polynomial. E.g., if the universe has 3 axes/variables like x,y,z in that (increasing) order, then a polynomial of level 2 at most mentions the first 2 variables, x and y, and definitely mentions second variable y but no "higher" variable like z .

0.14.66.1.6 CADCell A "cylindric algebraic cell" exists in an n-dimensional vector space called a "universe". A cell is a subspace of that universe with a possibly lower dimension and is "cylindric" only with respect to a specific variable ordering, numbered 0 to n-1. A cell is "algebraic", because its boundaries are represented by polynomials. Along each axis a cell is bound by either an non-empty open interval, called a "sector" and represented by two polynomials, or by a closed point-interval, called a "section" and represented by one polynomial. E.g. a section along axis k, said to be of "level k", requires a polynomial instead of a fixed number, because the fixed number can vary depending on the position along the (lower dimensional) axes 0 to k-1. This is why a section of level k is represented by a multivariate polynomial of "level k" (uses only the first k variables in the variable ordering). If we replace the first k-1 variables in that polynomial by specific numbers, we are left with a univariate polynomial whose root is then the fixed boundary number along axis k (if that polynomial has multiple roots, we also need to specify precisely which one of those represents the fixed boundary number). A different position, i.e., a different variable replacement, yields a possibly different univariate polynomial with a different root.

0.14.66.1.7 Level of a sector/section The number of the axis (with respect to some variable ordering that defines the same axis ordering) for which a sector/section defines the bounds. This number is also the level of the polynomial(s) inside the sector/section.

0.14.66.1.8 Open CADCell A cell is "open" if it only consists of sectors, i.e., it is open along all axes and is therefore a subspace with the same dimension as its universe.

0.14.66.2 Enumeration Type Documentation

0.14.66.2.1 ShrinkResult `enum smtrat::mcsat::onecellcad::recursive::ShrinkResult [strong]`

Enumerator

SUCCESS	
FAIL	

0.14.66.3 Function Documentation

0.14.66.3.1 operator<<() `std::ostream& smtrat::mcsat::onecellcad::recursive::operator<< ( std::ostream & os, const ShrinkResult & s ) [inline]`

0.14.66.3.2 setNullification() `void smtrat::mcsat::onecellcad::recursive::setNullification ( bool n ) [inline]`

0.14.66.4 Variable Documentation

0.14.66.4.1 cover_nullification `bool smtrat::mcsat::onecellcad::recursive::cover_nullification`

0.14.67 smtrat::mcsat::smtaf Namespace Reference

Data Structures

- struct AssignmentFinder
- struct DefaultSettings
- class AssignmentFinder_SMT

Typedefs

- using VariablePos = std::vector< carl::Variable >::const_iterator
- using VariableRange = std::pair< VariablePos, VariablePos >

Functions

- bool includes (const VariableRange &superset, const carl::Variables &subset)

0.14.67.1 Typedef Documentation

0.14.67.1.1 VariablePos

```
using smtrat::mcsat::smtaf::VariablePos = typedef std::vector<carl::↩
Variable>::const_iterator
```

0.14.67.1.2 VariableRange

```
using smtrat::mcsat::smtaf::VariableRange = typedef std::pair<VariablePos,VariablePo
```

0.14.67.2 Function Documentation

0.14.67.2.1 includes()

```
bool smtrat::mcsat::smtaf::includes (
    const VariableRange & superset,
    const carl::Variables & subset )  [inline]
```

0.14.68 smtrat::mcsat::variableordering Namespace Reference

Namespaces

- detail

Data Structures

- struct VariableIDs

Functions

- template<typename Constraints >
  std::vector< carl::Variable > feature_based (const Constraints &c)
- template<typename Constraints >
  std::vector< carl::Variable > feature_based_z3 (const Constraints &c)
- template<typename Constraints >
  std::vector< carl::Variable > feature_based_brown (const Constraints &c)

  According to https://www.usna.edu/Users/cs/wcbrown/research/ISSAC04/handout.pdf.
- template<typename Constraints >
  std::vector< carl::Variable > greedy_max_univariate (const Constraints &c)
- std::ostream & operator<< (std::ostream &os, const VariableIDs &vids)
- template<typename Constraints >
  void gatherVariables (carl::carlVariables &vars, const Constraints &constraints)

0.14.68.1 Function Documentation

0.14.68.1.1 feature_based()

```
template<typename Constraints >
std::vector<carl::Variable> smtrat::mcsat::variableordering::feature_based (
            const Constraints & c )
```

0.14.68.1.2 feature_based_brown()

```
template<typename Constraints >
std::vector<carl::Variable> smtrat::mcsat::variableordering::feature_based_brown (
            const Constraints & c )
```

According to https://www.usna.edu/Users/cs/wcbrown/research/ISSAC04/handout.↩pdf.

0.14.68.1.3 feature_based_z3()

```
template<typename Constraints >
std::vector<carl::Variable> smtrat::mcsat::variableordering::feature_based_z3 (
            const Constraints & c )
```

0.14.68.1.4 gatherVariables()

```
template<typename Constraints >
void smtrat::mcsat::variableordering::gatherVariables (
            carl::carlVariables & vars,
            const Constraints & constraints )
```

0.14.68.1.5 greedy_max_univariate()

```
template<typename Constraints >
std::vector<carl::Variable> smtrat::mcsat::variableordering::greedy_max_univariate (
            const Constraints & c )
```

0.14.68.1.6 operator<<()

```
std::ostream& smtrat::mcsat::variableordering::operator<< (
            std::ostream & os,
            const VariableIDs & vids )  [inline]
```

0.14.69 smtrat::mcsat::variableordering::detail Namespace Reference

Data Structures

- struct FeatureCollector

  This class manages features that are used to valuate variables on objects.

Functions

- template<typename Constraints , typename Calculator >
  double abstract_feature (const Constraints &constraints, double initial, std::function< double(double, double)> &&comb, Calculator &&calc)
- template<typename Constraints >
  double max_degree (const Constraints &constraints, carl::Variable v)

  The maximum degree of this variable.
- template<typename Constraints >
  double max_term_total_degree (const Constraints &constraints, carl::Variable v)

  The maximum total degree of a term involving this variable.
- template<typename Constraints >
  double max_coefficient (const Constraints &constraints, carl::Variable v)
- template<typename Constraints >
  double num_occurrences (const Constraints &constraints, carl::Variable v)
- carl::Bitset variablesOf (const ConstraintT &c, VariableIDs &vids)
- long countUnivariates (const std::vector< carl::Bitset > &constraints, std::size_t id)

- bool stillOccurs (const std::vector< carl::Bitset > &constraints, std::size_t id)
- carl::Variable findMax (const std::vector< carl::Bitset > &constraints, const VariableIDs &vids)
- void purgeVariable (std::vector< carl::Bitset > &constraints, carl::Variable v, const VariableIDs &vids)

0.14.69.1 Function Documentation

0.14.69.1.1 abstract_feature()

```
template<typename Constraints , typename Calculator >
double smtrat::mcsat::variableordering::detail::abstract_feature (
            const Constraints & constraints,
            double initial,
            std::function< double(double, double)> && comb,
            Calculator && calc )
```

0.14.69.1.2 countUnivariates()

```
long smtrat::mcsat::variableordering::detail::countUnivariates (
            const std::vector< carl::Bitset > & constraints,
            std::size_t id )  [inline]
```

0.14.69.1.3 findMax()

```
carl::Variable smtrat::mcsat::variableordering::detail::findMax (
            const std::vector< carl::Bitset > & constraints,
            const VariableIDs & vids )  [inline]
```

0.14.69.1.4 max_coefficient()

```
template<typename Constraints >
double smtrat::mcsat::variableordering::detail::max_coefficient (
            const Constraints & constraints,
            carl::Variable v )
```

0.14.69.1.5 max_degree()

```
template<typename Constraints >
double smtrat::mcsat::variableordering::detail::max_degree (
            const Constraints & constraints,
            carl::Variable v )
```

The maximum degree of this variable.

0.14.69.1.6 max_term_total_degree()

```
template<typename Constraints >
double smtrat::mcsat::variableordering::detail::max_term_total_degree (
            const Constraints & constraints,
            carl::Variable v )
```

The maximum total degree of a term involving this variable.

0.14.69.1.7 num_occurrences()

```
template<typename Constraints >
double smtrat::mcsat::variableordering::detail::num_occurrences (
            const Constraints & constraints,
            carl::Variable v )
```

0.14.69.1.8 purgeVariable()

```
void smtrat::mcsat::variableordering::detail::purgeVariable (
            std::vector< carl::Bitset > & constraints,
            carl::Variable v,
            const VariableIDs & vids )  [inline]
```

0.14.69.1.9 stillOccurs()

```
bool smtrat::mcsat::variableordering::detail::stillOccurs (
    const std::vector< carl::Bitset > & constraints,
    std::size_t id ) [inline]
```

0.14.69.1.10 variablesOf()

```
carl::Bitset smtrat::mcsat::variableordering::detail::variablesOf (
    const ConstraintT & c,
    VariableIDs & vids ) [inline]
```

0.14.70 smtrat::mcsat::vs Namespace Reference

Namespaces

- helper

Data Structures

- struct Explanation
- struct DefaultSettings
- class ExplanationGenerator

0.14.71 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)

0.14.71.1 Function Documentation

0.14.71.1.1 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.

0.14.71.1.2 generateTestCandidates()

```
static bool smtrat::mcsat::vs::helper::generateTest↩
Candidates (
            std::vector< TestCandidate > & results,
            const carl::Variable & eliminationVar,
            const FormulaSetT & constraints )  [static]
```

Generate all test candidates according to "vanilla" virtual substitution.
Returns false iff VS is not applicable.

0.14.71.1.3 generateZeros()

```
static bool smtrat::mcsat::vs::helper::generateZeros (
            const FormulaT & formula,
            const carl::Variable & eliminationVar,
            std::function< void(SqrtEx &&sqrtExpression, ConstraintsT &&sideConditions)>
yield_result )  [static]
```

Get zeros with side conditions of the given constraint.
Kind of a generator function. Passes generated zeros to a callback function to avoid copying.

0.14.71.1.4 getFormulaAtoms()

```
void smtrat::mcsat::vs::helper::getFormulaAtoms (
            const FormulaT & f,
            FormulaSetT & result )  [inline]
```

0.14.71.1.5 substitute()

```
bool smtrat::mcsat::vs::helper::substitute (
            const FormulaT & constr,
            const carl::Variable var,
            const carl::vs::Term< Poly > & term,
            FormulaT & result )  [inline]
```

0.14.71.1.6 to_formula()

```
FormulaT smtrat::mcsat::vs::helper::to_formula (
            const carl::vs::CaseDistinction< Poly > & docc )  [inline]
```

Converts a DisjunctionOfConstraintConjunctions to a regular Formula.

0.14.72 smtrat::onecellcad Namespace Reference

Namespaces

- recursive

0.14.73 smtrat::onecellcad::recursive Namespace Reference

Data Structures

- struct Section

  Represent a cell's closed-interval-boundary object along one single axis by an irreducible, multivariate polynomial of level k.
- struct Sector

  Represent a cell's open-interval boundary object along one single axis by two irreducible, multivariate polynomials of level k.

Typedefs

- using UniPoly = carl::UnivariatePolynomial< smtrat::Rational >
- using MultiPoly = carl::MultivariatePolynomial< smtrat::Rational >
- using MultiCoeffUniPoly = carl::UnivariatePolynomial< MultiPoly >
- using RANPoint = RealAlgebraicPoint< smtrat::Rational >
- using RANMap = std::map< carl::Variable, RAN >
- using OpenCADCell = std::vector< Sector >

  Represent a single open cell [1].

Functions

- std::ostream & operator<< (std::ostream &o, const Section &s)
- std::ostream & operator<< (std::ostream &o, const Sector &s)
- std::ostream & operator<< (std::ostream &o, const OpenCADCell &c)
- OpenCADCell createFullspaceCoveringCell (size_t level)
- size_t levelOf (const MultiPoly &poly, const std::vector< carl::Variable > &variableOrder)

  Find the index of the highest variable (wrt.
- std::map< carl::Variable, RAN > toStdMap (const RANPoint &point, size_t count, const std::vector< carl::↩ Variable > &variableOrder)

  Create a mapping from the first 'count'-many variables in the given 'variableOrder' to the first 'count'-many components of the given 'point'.
- std::optional< OpenCADCell > mergeCellWithPoly (OpenCADCell &cell, const RANPoint &point, const std↩ ::vector< carl::Variable > &variableOrder, const MultiPoly poly)

  Merge the given open OpenCADCell 'cell' that contains the given 'point' (called "alpha" in [1]) with a polynomial 'poly'.
- std::optional< OpenCADCell > createOpenCADCell (const std::vector< MultiPoly > polySet, const RANPoint &point, const std::vector< carl::Variable > &variableOrder)

  Construct a OpenCADCell for the given set of polynomials 'polySet' that contains the given 'point'.

0.14.73.1 Typedef Documentation

0.14.73.1.1 MultiCoeffUniPoly

`using smtrat::onecellcad::recursive::MultiCoeffUniPoly = typedef carl::UnivariatePolynomial<MultiPoly>`

0.14.73.1.2 MultiPoly

`using smtrat::onecellcad::recursive::MultiPoly = typedef carl::Multivariate↩ Polynomial<smtrat::Rational>`

0.14.73.1.3 OpenCADCell

`using smtrat::onecellcad::recursive::OpenCADCell = typedef std↩ ::vector<Sector>`

Represent a single open cell [1].
A cell is a collection of boundary objects along each axis. In case of an open cell, the boundary objects are all sectors.

0.14.73.1.4 RANMap

`using smtrat::onecellcad::recursive::RANMap = typedef std::map<carl::↩ Variable, RAN>`

0.14.73.1.5 RANPoint

`using smtrat::onecellcad::recursive::RANPoint = typedef RealAlgebraic↩ Point<smtrat::Rational>`

0.14.73.1.6 UniPoly

`using smtrat::onecellcad::recursive::UniPoly = typedef carl::Univariate↩ Polynomial<smtrat::Rational>`

0.14.73.2 Function Documentation

0.14.73.2.1 createFullspaceCoveringCell()

```
OpenCADCell smtrat::onecellcad::recursive::create↩ FullspaceCoveringCell (
            size_t level )
```

0.14.73.2.2 createOpenCADCell()

```
std::optional<OpenCADCell> smtrat::onecellcad::recursive↩
::createOpenCADCell (
            const std::vector< MultiPoly > polySet,
            const RANPoint & point,
            const std::vector< carl::Variable > & variableOrder )
```

Construct a OpenCADCell for the given set of polynomials 'polySet' that contains the given 'point'.
The returned cell represents the largest region that is sign-invariant on all polynomials in the 'polySet' and is cylindrical with respect to the variables ordering given in 'variableOrder'. Note that this construction is only correct if plugging in the 'point' into any polynomial of the 'polySet' yields a non-zero value, i.e. 'point' is not a root of any polynomial in 'polySet', otherwise no OpenCADCell is returned. Note that the dimension (number of components) of the 'point' == the number of variables in 'variableOrder' and that any polynomial of the 'polySet' must be irreducible and mention only variables from 'variableOrder'.

0.14.73.2.3 levelOf()

```
size_t smtrat::onecellcad::recursive::levelOf (
            const MultiPoly & poly,
            const std::vector< carl::Variable > & variableOrder )
```

Find the index of the highest variable (wrt.
the ordering in 'variableOrder') that occurs with positive degree in 'poly'. Since levelOf is a math concept it starts counting at 1! Examples:

- levelOf(2) == 0 wrt. any variable order
- levelOf(0∗x+2) == 0 wrt. any variable order
- levelOf(x+2) == 1 wrt. [x < y < z]
- levelOf(x+2) == 2 wrt. [y < x < z]
- levelOf(x+2) == 3 wrt. [y < z < x]
- levelOf(x∗y+2) == 2 wrt. [x < y < z] because of y
- levelOf(x∗y+2) == 2 wrt. [y < x < z] because of x
- levelOf(x∗y+2) == 3 wrt. [x < z < y] because of y Preconditions:
- 'variables(poly)' must be a subset of 'variableOrder'.

0.14.73.2.4 mergeCellWithPoly()

```
std::optional<OpenCADCell> smtrat::onecellcad::recursive↩
::mergeCellWithPoly (
            OpenCADCell & cell,
            const RANPoint & point,
            const std::vector< carl::Variable > & variableOrder,
            const MultiPoly poly )
```

Merge the given open OpenCADCell 'cell' that contains the given 'point' (called "alpha" in [1]) with a polynomial 'poly'.
Before the merge 'cell' represents a region that is sign-invariant on other (previously merged) polynomials (all signs are non-zero). The returned cell represents a region that is additionally sign-invariant on 'poly' (also with non-zero sign).

Returns

either an OpenCADCell or nothing (representing a failed construction)

0.14.73.2.5 operator<<() [1/3]

```
std::ostream& smtrat::onecellcad::recursive::operator<< (
            std::ostream & o,
            const OpenCADCell & c )
```

0.14.73.2.6 operator<<() [2/3]

```
std::ostream& smtrat::onecellcad::recursive::operator<< (
    std::ostream & o,
    const Section & s )
```

0.14.73.2.7 operator<<() [3/3]

```
std::ostream& smtrat::onecellcad::recursive::operator<< (
    std::ostream & o,
    const Sector & s )
```

0.14.73.2.8 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'.

0.14.74 smtrat::options_detail Namespace Reference

Functions

- void print_cmake_options ()
- void print_info ()
- void print_license ()
- void print_version ()

0.14.74.1 Function Documentation

0.14.74.1.1 print_cmake_options()

```
void smtrat::options_detail::print_cmake_options ( )
```

0.14.74.1.2 print_info()

```
void smtrat::options_detail::print_info ( )
```

0.14.74.1.3 print_license()

```
void smtrat::options_detail::print_license ( )
```

0.14.74.1.4 print_version()

```
void smtrat::options_detail::print_version ( )
```

0.14.75 smtrat::parseformula Namespace Reference

Data Structures

- class FormulaCollector

0.14.76 smtrat::parser Namespace Reference

Namespaces

- arithmetic
- conversion
- core
- types
- uninterpreted

Data Structures

- struct AttributeValueParser
- struct AttributeParser
- struct Skipper
- struct IdentifierParser
- class OutputWrapper
- class InstructionHandler
- struct RationalPolicies

  Specialization of qi::real_policies for a Rational.
- struct NumeralParser

  Parses numerals: `(0 | [1-9][0-9]*)`
- struct DecimalParser

  Parses decimals: `numeral.0*numeral`
- struct HexadecimalParser

  Parses hexadecimals: `#x[0-9a-fA-F]+`
- struct BinaryParser

  Parses binaries: `#b[01]+`
- struct StringParser

  Parses strings: `".+"` *with escape sequences* `\\"` *and* `\\\\`
- struct SimpleSymbolParser

  Parses symbols: `simple_symbol | quoted_symbol` *where.*
- struct SymbolParser
- struct KeywordParser

  Parses keywords: `:simple_symbol`
- class SMTLIBParser
- struct ParserSettings
- struct LogicParser
- struct ErrorHandler
- struct QuantifierParser
- struct QEParser
- struct ScriptParser
- struct SpecConstantParser
- struct SExpressionParser
- struct SortParser
- struct QualifiedIdentifierParser
- struct SortedVariableParser
- struct TermParser
- struct AbstractTheory

  Base class for all theories.
- struct ArithmeticTheory

  Implements the theory of arithmetic, including LRA, LIA, NRA and NIA.
- class Attribute

  Represents an Attribute.
- struct BitvectorTheory

  Implements the theory of bitvectors.
- struct BooleanEncodingTheory

  Implements the theory of bitvectors.
- struct TheoryError
- struct Identifier
- struct SExpressionSequence
- struct CoreTheory

  Implements the core theory of the booleans.

- struct FunctionInstantiator
- struct IndexedFunctionInstantiator
- struct Instantiator
- struct UserFunctionInstantiator
- struct ParserState
- struct Theories

  The Theories class combines all implemented theories and provides a single interface to interact with all theories at once.
- struct FixedWidthConstant

  Represents a constant of a fixed width.
- struct UninterpretedTheory

  Implements the theory of equalities and uninterpreted functions.

Typedefs

- typedef boost::spirit::istream_iterator 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 class OptimizationType { Maximize , Minimize }

Functions

- std::ostream & operator<< (std::ostream &os, OptimizationType ot)
- template<typename T >
  void registerParserSettings (T &parser)
- std::ostream & operator<< (std::ostream &os, const Attribute &attr)
- void at_most_k_helper (size_t k, const std::vector< FormulaT > &set, FormulasT &working, long long int prev_idx, FormulasT &results)
- FormulaT at_most_k (size_t k, const std::vector< FormulaT > &set)
- std::ostream & operator<< (std::ostream &os, const Identifier &identifier)
- template<typename T >
  std::ostream & operator<< (std::ostream &os, const SExpressionSequence< T > &ses)
- template<typename T >
  std::ostream & operator<< (std::ostream &os, const FixedWidthConstant< T > &fwc)

0.14.76.1 Typedef Documentation

0.14.76.1.1 BaseIteratorType `typedef boost::spirit::istream_iterator smtrat::parser::BaseIteratorType`

0.14.76.1.2 Iterator `typedef PositionIteratorType smtrat::parser::Iterator`

0.14.76.1.3 PositionIteratorType `typedef boost::spirit::line_pos_iterator<BaseIteratorType> smtrat::parser::PositionIteratorType`

0.14.76.1.4 SExpression `template<typename T >`

```
using smtrat::parser::SExpression = typedef boost::variant<T, boost::recursive_wrapper<SExpressionSequence<T>
>>
```

0.14.76.2 Enumeration Type Documentation

0.14.76.2.1 OptimizationType `enum smtrat::parser::OptimizationType [strong]`

Enumerator

Maximize	
Minimize	

0.14.76.3 Function Documentation

0.14.76.3.1 at_most_k()

```
FormulaT smtrat::parser::at_most_k (
    size_t k,
    const std::vector< FormulaT > & set )
```

0.14.76.3.2 at_most_k_helper()

```
void smtrat::parser::at_most_k_helper (
    size_t k,
    const std::vector< FormulaT > & set,
    FormulasT & working,
    long long int prev_idx,
    FormulasT & results )
```

0.14.76.3.3 operator<<() [1/5]

```
std::ostream& smtrat::parser::operator<< (
    std::ostream & os,
    const Attribute & attr ) [inline]
```

0.14.76.3.4 operator<<() [2/5]

```
template<typename T >
std::ostream& smtrat::parser::operator<< (
    std::ostream & os,
    const FixedWidthConstant< T > & fwc ) [inline]
```

0.14.76.3.5 operator<<() [3/5]

```
std::ostream& smtrat::parser::operator<< (
    std::ostream & os,
    const Identifier & identifier ) [inline]
```

0.14.76.3.6 operator<<() [4/5]

```
template<typename T >
std::ostream& smtrat::parser::operator<< (
    std::ostream & os,
    const SExpressionSequence< T > & ses ) [inline]
```

0.14.76.3.7 operator<<() [5/5]

```
std::ostream& smtrat::parser::operator<< (
    std::ostream & os,
    OptimizationType ot ) [inline]
```

0.14.76.3.8 registerParserSettings()

```
template<typename T >
void smtrat::parser::registerParserSettings (
            T & parser )
```

0.14.77 smtrat::parser::arithmetic Namespace Reference

Typedefs

- using OperatorType = boost::variant< Poly::ConstructorOperation, carl::Relation >

Functions

- FormulaT makeConstraint (ArithmeticTheory &at, const Poly &lhs, const Poly &rhs, carl::Relation rel)
- bool isBooleanIdentity (const OperatorType &op, const std::vector< types::TermType > &arguments, TheoryError &errors)

0.14.77.1 Typedef Documentation

0.14.77.1.1 OperatorType

```
using smtrat::parser::arithmetic::OperatorType = typedef boost↩
::variant<Poly::ConstructorOperation, carl::Relation>
```

0.14.77.2 Function Documentation

0.14.77.2.1 isBooleanIdentity()

```
bool smtrat::parser::arithmetic::isBooleanIdentity (
            const OperatorType & op,
            const std::vector< types::TermType > & arguments,
            TheoryError & errors )  [inline]
```

0.14.77.2.2 makeConstraint()

```
FormulaT smtrat::parser::arithmetic::makeConstraint (
            ArithmeticTheory & at,
            const Poly & lhs,
            const Poly & rhs,
            carl::Relation rel )  [inline]
```

0.14.78 smtrat::parser::conversion Namespace Reference

Data Structures

- struct Converter
- struct Converter< types::BVTerm >
- struct Converter< Poly >
- struct Converter< FormulaT >
- struct VariantConverter

  Converts variants to some type using the Converter class.
- struct VariantVariantConverter

  Converts variants to another variant type not using the Converter class.
- struct VectorVariantConverter

  Converts a vector of variants to a vector of some type using the Converter class.

0.14.79 smtrat::parser::core Namespace Reference

Functions

- bool convertTerm (const types::TermType &term, FormulaT &result)
- bool convertArguments (const std::vector< types::TermType > &arguments, std::vector< FormulaT > &result, TheoryError &errors)

0.14.79.1 Function Documentation

0.14.79.1.1 convertArguments()

```
bool smtrat::parser::core::convertArguments (
    const std::vector< types::TermType > & arguments,
    std::vector< FormulaT > & result,
    TheoryError & errors ) [inline]
```

0.14.79.1.2 convertTerm()

```
bool smtrat::parser::core::convertTerm (
    const types::TermType & term,
    FormulaT & result ) [inline]
```

0.14.80 smtrat::parser::types Namespace Reference

Data Structures

- struct CoreTheory

  Types of the core theory.
- struct ArithmeticTheory

  Types of the arithmetic theory.
- struct BitvectorTheory

  Types of the theory of bitvectors.
- struct UninterpretedTheory

  Types of the theory of equalities and uninterpreted functions.

Typedefs

- typedef carl::BVTerm BVTerm

  Typedef for bitvector term.
- typedef carl::BVVariable BVVariable
- typedef carl::BVConstraint BVConstraint

  Typedef for bitvector constraint.
- using UTerm = carl::UTerm
- typedef carl::mpl_concatenate< CoreTheory::ConstTypes, >::type ConstTypes

  List of all types of constants.
- typedef carl::mpl_variant_of< ConstTypes >::type ConstType

  Variant type for all constants.
- typedef carl::mpl_concatenate< CoreTheory::VariableTypes, >::type VariableTypes

  List of all types of variables.
- typedef carl::mpl_variant_of< VariableTypes >::type VariableType

  Variant type for all variables.
- typedef carl::mpl_concatenate< CoreTheory::ExpressionTypes, >::type ExpressionTypes

  List of all types of expressions.
- typedef carl::mpl_variant_of< ExpressionTypes >::type ExpressionType

  Variant type for all expressions.

- typedef carl::mpl_concatenate< CoreTheory::TermTypes, >::type TermTypes

  List of all types of terms.
- typedef carl::mpl_variant_of< TermTypes >::type TermType

  Variant type for all terms.
- typedef carl::mpl_concatenate< TermTypes, boost::mpl::vector< SExpressionSequence< types::ConstType >, std::string, bool, boost::spirit::unused_type > >::type AttributeTypes

  List of all types of attributes.
- typedef carl::mpl_variant_of< AttributeTypes >::type AttributeValue

  Variant type for all attributes.

0.14.80.1 Typedef Documentation

0.14.80.1.1 AttributeTypes

`typedef carl::mpl_concatenate< TermTypes, boost::mpl::vector< SExpressionSequence<t std::string, bool, boost::spirit::unused_type > >::type smtrat::parser::types::AttributeTypes`

List of all types of attributes.

0.14.80.1.2 AttributeValue

`typedef carl::mpl_variant_of<AttributeTypes>::type smtrat::parser::types::Attribute`

Variant type for all attributes.

0.14.80.1.3 BVConstraint

`typedef carl::BVConstraint smtrat::parser::types::BVConstraint`

Typedef for bitvector constraint.

0.14.80.1.4 BVTerm

`typedef carl::BVTerm smtrat::parser::types::BVTerm`

Typedef for bitvector term.

0.14.80.1.5 BVVariable

`typedef carl::BVVariable smtrat::parser::types::BVVariable`

0.14.80.1.6 ConstType

`typedef carl::mpl_variant_of<ConstTypes>::type smtrat::parser::types::ConstType`

Variant type for all constants.

0.14.80.1.7 ConstTypes

`typedef carl::mpl_concatenate< CoreTheory::ConstTypes, >::type smtrat::parser::types:`

List of all types of constants.

0.14.80.1.8 ExpressionType

`typedef carl::mpl_variant_of<ExpressionTypes>::type smtrat::parser::types::Express`

Variant type for all expressions.

0.14.80.1.9 ExpressionTypes

`typedef carl::mpl_concatenate< CoreTheory::ExpressionTypes, >↩ ::type smtrat::parser::types::ExpressionTypes`

List of all types of expressions.

0.14.80.1.10 TermType

`typedef carl::mpl_variant_of<TermTypes>::type smtrat::parser::types::TermType`

Variant type for all terms.

0.14.80.1.11 TermTypes `typedef carl::mpl_concatenate< CoreTheory::TermTypes, >::type smtrat::parser::types::`
List of all types of terms.

0.14.80.1.12 UTerm `using smtrat::parser::types::UTerm = typedef carl::UTerm`

0.14.80.1.13 VariableType `typedef carl::mpl_variant_of<VariableTypes>::type smtrat::parser::types::VariableTyp`
Variant type for all variables.

0.14.80.1.14 VariableTypes `typedef carl::mpl_concatenate< CoreTheory::VariableTypes, >::type smtrat::parser::types::VariableTypes`
List of all types of variables.

0.14.81 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)

0.14.81.1 Function Documentation

0.14.81.1.1 convertArguments()
```
bool smtrat::parser::uninterpreted::convertArguments (
    const std::vector< types::TermType > & arguments,
    std::vector< types::UTerm > & result,
    TheoryError & errors ) [inline]
```

0.14.81.1.2 convertTerm()
```
bool smtrat::parser::uninterpreted::convertTerm (
    const types::TermType & term,
    types::UTerm & result ) [inline]
```

0.14.82 smtrat::qe Namespace Reference

Namespaces

- cad
- fm

Typedefs

- using QEQuery = std::vector< std::pair< QuantifierType, std::vector< carl::Variable > >>

Enumerations

- enum class QuantifierType { EXISTS , FORALL }

Functions

- std::ostream & operator<< (std::ostream &os, QuantifierType qt)
- std::ostream & operator<< (std::ostream &os, const QEQuery &q)
- std::vector< std::pair< QuantifierType, carl::Variable > > flattenQEQuery (const QEQuery &query)
- FormulaT eliminateQuantifiers (const FormulaT &qfree, const QEQuery &quantifiers)

0.14.82.1 Typedef Documentation

0.14.82.1.1 QEQuery

```
using smtrat::qe::QEQuery = typedef std::vector<std::pair<QuantifierType,std↩
::vector<carl::Variable> >>
```

0.14.82.2 Enumeration Type Documentation

0.14.82.2.1 QuantifierType

```
enum smtrat::qe::QuantifierType [strong]
```

Enumerator

EXISTS	
FORALL	

0.14.82.3 Function Documentation

0.14.82.3.1 eliminateQuantifiers()

```
FormulaT smtrat::qe::eliminateQuantifiers (
    const FormulaT & qfree,
    const QEQuery & quantifiers ) [inline]
```

0.14.82.3.2 flattenQEQuery()

```
std::vector<std::pair<QuantifierType,carl::Variable> > smtrat↩
::qe::flattenQEQuery (
    const QEQuery & query ) [inline]
```

0.14.82.3.3 operator<<() [1/2]

```
std::ostream& smtrat::qe::operator<< (
    std::ostream & os,
    const QEQuery & q ) [inline]
```

0.14.82.3.4 operator<<() [2/2]

```
std::ostream& smtrat::qe::operator<< (
    std::ostream & os,
    QuantifierType qt ) [inline]
```

0.14.83 smtrat::qe::cad Namespace Reference

Data Structures

- class CAD
- struct CADSettings
- class CADElimination
- class Projection

Functions

- template<typename key , typename value >
  std::multimap< value, key > flip_map (const std::map< key, value > &src)
- template<typename S >
  std::ostream & operator<< (std::ostream &os, const Projection< S > &p)
- FormulaT eliminateQuantifiers (const FormulaT &qfree, const QEQuery &quantifiers)

0.14.83.1 Function Documentation

0.14.83.1.1 eliminateQuantifiers()

```
FormulaT smtrat::qe::cad::eliminateQuantifiers (
    const FormulaT & qfree,
    const QEQuery & quantifiers )
```

0.14.83.1.2 flip_map()

```
template<typename key , typename value >
std::multimap<value, key> smtrat::qe::cad::flip_map (
    const std::map< key, value > & src )
```

0.14.83.1.3 operator<<()

```
template<typename S >
std::ostream& smtrat::qe::cad::operator<< (
    std::ostream & os,
    const Projection< S > & p )
```

0.14.84 smtrat::qe::fm Namespace Reference

Data Structures

- class FourierMotzkinQE

  Provides a simple implementation for Fourier Motzkin variable elimination for linear, existentially quantified constraints.

Functions

- FormulaT eliminateQuantifiers (const FormulaT &qfree, const QEQuery &quantifiers)

0.14.84.1 Function Documentation

0.14.84.1.1 eliminateQuantifiers()

```
FormulaT smtrat::qe::fm::eliminateQuantifiers (
    const FormulaT & qfree,
    const QEQuery & quantifiers )
```

0.14.85 smtrat::resource Namespace Reference

Data Structures

- class Limiter

Functions

- void setCPULimit (std::size_t)
- std::size_t getCPULimit ()
- std::size_t usedCPU ()
- void setMemoryLimit (std::size_t)
- std::size_t getMemoryLimit ()
- void installSignalHandler ()
- void signalHandler (int signal)

0.14.85.1 Function Documentation

0.14.85.1.1 getCPULimit() `std::size_t smtrat::resource::getCPULimit ( ) [inline]`

0.14.85.1.2 getMemoryLimit() `std::size_t smtrat::resource::getMemoryLimit ( ) [inline]`

0.14.85.1.3 installSignalHandler() `void smtrat::resource::installSignalHandler ( ) [inline]`

0.14.85.1.4 setCPULimit()

```
void smtrat::resource::setCPULimit (
    std::size_t ) [inline]
```

0.14.85.1.5 setMemoryLimit()

```
void smtrat::resource::setMemoryLimit (
    std::size_t ) [inline]
```

0.14.85.1.6 signalHandler()

```
void smtrat::resource::signalHandler (
    int signal ) [inline]
```

0.14.85.1.7 usedCPU() `std::size_t smtrat::resource::usedCPU ( ) [inline]`

0.14.86 smtrat::sat Namespace Reference

Namespaces

- detail

0.14.87 smtrat::sat::detail Namespace Reference

Data Structures

- struct ClauseChecker

Functions

- template<typename T >
  void validateClause (const T &t, bool enabled)
- template<typename T , typename VM , typename BCM >
  void validateClause (const T &t, const VM &vm, const BCM &bcm, bool enabled)

0.14.87.1 Function Documentation

0.14.87.1.1 validateClause() [1/2] template<typename T >

```
void smtrat::sat::detail::validateClause (
    const T & t,
    bool enabled )
```

0.14.87.1.2 validateClause() [2/2] template<typename T , typename VM , typename BCM >

```
void smtrat::sat::detail::validateClause (
    const T & t,
    const VM & vm,
    const BCM & bcm,
    bool enabled )
```

0.14.88 smtrat::settings Namespace Reference

Data Structures

- struct CoreSettings
- struct SolverSettings
- struct ModuleSettings
- struct Settings

0.14.89 smtrat::statistics Namespace Reference

Data Structures

- struct StatisticsSettings

Functions

- template<typename T >
  void registerStatisticsSettings (T &parser)

0.14.89.1 Function Documentation

0.14.89.1.1 registerStatisticsSettings()

```
template<typename T >
void smtrat::statistics::registerStatisticsSettings (
            T & parser )
```

0.14.90 smtrat::subtropical Namespace Reference

Implements data structures and encodings for the subtropical method.

Data Structures

- struct Moment

  Represents the normal vector component and the sign variable assigned to a variable in an original constraint.
- struct Vertex

  Represents a term of an original constraint and assigns him a rating variable if a weak separator is searched.
- struct Separator

  Represents the class of all original constraints with the same left hand side after a normalization.
- class Encoding

Enumerations

- enum class SeparatorType { STRICT = 0 , WEAK = 1 , SEMIWEAK = 2 }
- enum class Direction { BOTH = 0 , NEGATIVE = 1 , POSITIVE = 2 }

  Subdivides the relations into classes with the same linearization result.

Functions

- carl::BasicConstraint< Poly > normalize (const carl::BasicConstraint< Poly > &c)
- std::optional< Direction > direction (carl::Relation r)
- std::optional< Direction > direction_for_equality (const Separator &s)
- FormulaT transform_to_equation (const FormulaT &formula)

  Requires a quantifier-free real arithmetic formula with no negations.
- FormulaT encode_as_formula (const FormulaT &formula, Encoding &encoding, SeparatorType separator_type)

  Requires a quantifier-free real arithmetic formula with no negations.

- FormulaT encode_as_formula_alt (const FormulaT &formula, Encoding &encoding, SeparatorType separator_type)

  Requires a quantifier-free real arithmetic formula with no negations.

0.14.90.1 Detailed Description

Implements data structures and encodings for the subtropical method.

0.14.90.2 Enumeration Type Documentation

0.14.90.2.1 Direction `enum smtrat::subtropical::Direction [strong]`

Subdivides the relations into classes with the same linearization result.

Enumerator

BOTH	
NEGATIVE	
POSITIVE	

0.14.90.2.2 SeparatorType `enum smtrat::subtropical::SeparatorType [strong]`

Enumerator

STRICT	
WEAK	
SEMIWEAK	

0.14.90.3 Function Documentation

0.14.90.3.1 direction()

```
std::optional<Direction> smtrat::subtropical::direction (
    carl::Relation r ) [inline]
```

0.14.90.3.2 direction_for_equality()

```
std::optional<Direction> smtrat::subtropical::direction_↩
for_equality (
    const Separator & s ) [inline]
```

0.14.90.3.3 encode_as_formula()

```
FormulaT smtrat::subtropical::encode_as_formula (
    const FormulaT & formula,
    Encoding & encoding,
    SeparatorType separator_type ) [inline]
```

Requires a quantifier-free real arithmetic formula with no negations.

Parameters

formula	The formula to translate

Returns

linear formula

0.14.90.3.4 encode_as_formula_alt()

```
FormulaT smtrat::subtropical::encode_as_formula_alt (
    const FormulaT & formula,
    Encoding & encoding,
    SeparatorType separator_type ) [inline]
```

Requires a quantifier-free real arithmetic formula with no negations.

Parameters

formula	The formula to translate

Returns

linear formula

0.14.90.3.5 normalize()

```
carl::BasicConstraint<Poly> smtrat::subtropical::normalize (
    const carl::BasicConstraint< Poly > & c ) [inline]
```

0.14.90.3.6 transform_to_equation()

```
FormulaT smtrat::subtropical::transform_to_equation (
    const FormulaT & formula ) [inline]
```

Requires a quantifier-free real arithmetic formula with no negations.

Parameters

formula	The formula to transform

Returns

an equisatisfiable equation

0.14.91 smtrat::unsatcore Namespace Reference

Contains strategy implementations for unsat core computations.

Data Structures

- class UnsatCoreBackend
- class UnsatCoreBackend< Solver, UnsatCoreStrategy::ModelExclusion >

  Implements an easy strategy to obtain an unsatisfiable core.

0.14.91.1 Detailed Description

Contains strategy implementations for unsat core computations.

0.14.92 smtrat::validation Namespace Reference

Data Structures

- class ValidationCollector
- class ValidationPoint
- struct ValidationPrinter
- struct ValidationSettings

Enumerations

- enum class ValidationOutputFormat { SMTLIB }

Functions

- auto & get (const std::string &channel, const std::string &file, int line)
- template<ValidationOutputFormat SOF>
  std::ostream & operator<< (std::ostream &os, ValidationPrinter< SOF >)
- template<> std::ostream & operator<< (std::ostream &os, ValidationPrinter< ValidationOutputFormat::SMTLIB >)
- auto validation_formulas_as_smtlib ()
- void validation_formulas_to_smtlib_file (const std::string &filename)
- template<typename T >
  void registerValidationSettings (T &parser)

0.14.92.1 Enumeration Type Documentation

0.14.92.1.1 ValidationOutputFormat `enum smtrat::validation::ValidationOutputFormat [strong]`

Enumerator

SMTLIB

0.14.92.2 Function Documentation

0.14.92.2.1 get()

```
auto& smtrat::validation::get (
    const std::string & channel,
    const std::string & file,
    int line ) [inline]
```

0.14.92.2.2 operator<<() [1/2]

```
template<ValidationOutputFormat SOF>
std::ostream& smtrat::validation::operator<< (
    std::ostream & os,
    ValidationPrinter< SOF > )
```

0.14.92.2.3 operator<<() [2/2]

```
template<>
std::ostream& smtrat::validation::operator<< (
    std::ostream & os,
    ValidationPrinter< ValidationOutputFormat::SMTLIB > )
```

0.14.92.2.4 registerValidationSettings()

```
template<typename T >
void smtrat::validation::registerValidationSettings (
    T & parser )
```

0.14.92.2.5 validation_formulas_as_smtlib()

```
auto smtrat::validation::validation_formulas_as_↩
smtlib ( )
```

0.14.92.2.6 validation_formulas_to_smtlib_file()

```
void smtrat::validation::validation_formulas_to_↩
smtlib_file (
            const std::string & filename )
```

0.14.93 smtrat::vb Namespace Reference

Data Structures

- class Variable

  Class for a variable.
- class Bound

  Class for the bound of a variable.
- class VariableBounds

  Class to manage the bounds of a variable.

Functions

- template<typename Type >
  std::ostream & operator<< (std::ostream &_os, const VariableBounds< Type > &_vs)

  Prints the contents of the given variable bounds manager to the given stream.

0.14.93.1 Function Documentation

0.14.93.1.1 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.

0.14.94 smtrat::vs Namespace Reference

Data Structures

- class Condition
- struct unsignedTripleCmp
- class State
- class Substitution
- struct substitutionPointerEqual
- struct substitutionPointerHash

Typedefs

- typedef std::set< carl::PointerSet< Condition > > ConditionSetSet
- typedef std::set< ConditionSetSet > ConditionSetSetSet
- typedef std::list< const Condition * > ConditionList
- typedef std::vector< ConditionList > DisjunctionOfConditionConjunctions

- typedef std::pair< size_t, std::pair< size_t, size_t > > UnsignedTriple
- typedef std::pair< UnsignedTriple, vs::State * > ValStatePair
- typedef std::map< UnsignedTriple, vs::State *, unsignedTripleCmp > ValuationMap
- typedef std::vector< smtrat::ConstraintT > ConstraintVector

  a vector of constraints
- typedef std::vector< ConstraintVector > DisjunctionOfConstraintConjunctions

  a vector of vectors of constraints
- template<typename T >
  using SubstitutionFastPointerMap = std::unordered_map< const 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 &_conflicting↩ Variables, 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, DisjunctionOfConstraintConjunctio &_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.

0.14.94.1 Typedef Documentation

0.14.94.1.1 ConditionList

```
typedef std::list< const Condition* > smtrat::vs::ConditionList
```

0.14.94.1.2 ConditionSetSet

```
typedef std::set< carl::PointerSet<Condition> > smtrat::vs::ConditionSetSet
```

0.14.94.1.3 ConditionSetSetSet

```
typedef std::set< ConditionSetSet > smtrat::vs::ConditionSetSetSet
```

0.14.94.1.4 ConstraintVector

```
typedef std::vector<smtrat::ConstraintT> smtrat::vs::ConstraintVector
```

a vector of constraints

0.14.94.1.5 DisjunctionOfConditionConjunctions

```
typedef std::vector< ConditionList > smtrat::vs::DisjunctionOfC
```

0.14.94.1.6 DisjunctionOfConstraintConjunctions

```
typedef std::vector<ConstraintVector> smtrat::vs::DisjunctionO
```

a vector of vectors of constraints

0.14.94.1.7 SubstitutionFastPointerMap

```
template<typename T >
using smtrat::vs::SubstitutionFastPointerMap = typedef std::unordered_map<const smtrat::Poly*,
T, substitutionPointerHash, substitutionPointerEqual>
```

0.14.94.1.8 UnsignedTriple

```
typedef std::pair<size_t, std::pair<size_t, size_t> > smtrat::vs::UnsignedTriple
```

0.14.94.1.9 ValStatePair

```
typedef std::pair<UnsignedTriple, vs::State*> smtrat::vs::ValStatePair
```

0.14.94.1.10 ValuationMap

```
typedef std::map<UnsignedTriple, vs::State*, unsignedTripleCmp>
smtrat::vs::ValuationMap
```

0.14.94.2 Function Documentation

0.14.94.2.1 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.

0.14.94.2.2 getEvenBitStrings() [1/2]

```
void smtrat::vs::getEvenBitStrings (
    size_t _length,
    std::vector< std::bitset< MAX_PRODUCT_SPLIT_NUMBER > > & _strings )
```

Parameters

_length	The maximal length of the bit strings with even parity to compute.
_strings	All bit strings of length less or equal the given length with even parity.

0.14.94.2.3 getEvenBitStrings() [2/2]

```
void smtrat::vs::getEvenBitStrings (
    std::size_t _length,
    std::vector< std::bitset< MAX_PRODUCT_SPLIT_NUMBER > > & _strings )
```

0.14.94.2.4 getOddBitStrings() [1/2]

```
void smtrat::vs::getOddBitStrings (
    size_t _length,
    std::vector< std::bitset< MAX_PRODUCT_SPLIT_NUMBER > > & _strings )
```

Parameters

_length	The maximal length of the bit strings with odd parity to compute.
_strings	All bit strings of length less or equal the given length with odd parity.

0.14.94.2.5 getOddBitStrings() [2/2]

```
void smtrat::vs::getOddBitStrings (
    std::size_t _length,
    std::vector< std::bitset< MAX_PRODUCT_SPLIT_NUMBER > > & _strings )
```

0.14.94.2.6 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.

0.14.94.2.7 operator<<() [1/2]

```
std::ostream& smtrat::vs::operator<< (
    std::ostream & _out,
    const Condition & _condition )
```

0.14.94.2.8 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.

0.14.94.2.9 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.

0.14.94.2.10 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.

0.14.94.2.11 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	

0.14.94.2.12 simplify() [3/3]

```
void smtrat::vs::simplify (
    DisjunctionOfConstraintConjunctions & _toSimplify,
    Variables & _conflictingVars,
    const smtrat::EvalDoubleIntervalMap & _solutionSpace )
```

0.14.94.2.13 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.

0.14.94.2.14 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.

0.14.94.2.15 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.

0.14.94.2.16 splitSosDecompositions()

```
void smtrat::vs::splitSosDecompositions (
DisjunctionOfConstraintConjunctions & _toSimplify )
```

0.14.94.2.17 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.

0.14.94.2.18 substitute() [2/2]

```
bool smtrat::vs::substitute (
const smtrat::ConstraintT & _cons,
const Substitution & _subs,
DisjunctionOfConstraintConjunctions & _result,
bool _accordingPaper,
Variables & _conflictingVariables,
const smtrat::EvalDoubleIntervalMap & _solutionSpace )
```

0.14.94.2.19 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.

0.14.94.2.20 substituteEpsGradients() [2/2]

```
bool smtrat::vs::substituteEpsGradients (
    const smtrat::ConstraintT & _cons,
    const Substitution & _subs,
    const Relation _relation,
    DisjunctionOfConstraintConjunctions & _result,
    bool _accordingPaper,
    Variables & _conflictingVariables,
    const smtrat::EvalDoubleIntervalMap & _solutionSpace )
```

0.14.94.2.21 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.

0.14.94.2.22 substituteInf() [2/2]

```
void smtrat::vs::substituteInf (
    const smtrat::ConstraintT & _cons,
    const Substitution & _subs,
    DisjunctionOfConstraintConjunctions & _result,
    Variables & _conflictingVariables,
```

```
const smtrat::EvalDoubleIntervalMap & _solutionSpace )
```

0.14.94.2.23 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.

0.14.94.2.24 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.

0.14.94.2.25 substituteNormal() [2/2]

```
bool smtrat::vs::substituteNormal (
    const smtrat::ConstraintT & _cons,
    const Substitution & _subs,
    DisjunctionOfConstraintConjunctions & _result,
    bool _accordingPaper,
    Variables & _conflictingVariables,
    const smtrat::EvalDoubleIntervalMap & _solutionSpace )
```

0.14.94.2.26 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).

0.14.94.2.27 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).

0.14.94.2.28 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).

0.14.94.2.29 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).

0.14.94.2.30 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.

0.14.94.2.31 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.

0.14.94.2.32 substitutePlusEps() [2/2]

```
bool smtrat::vs::substitutePlusEps (
    const smtrat::ConstraintT & _cons,
    const Substitution & _subs,
    DisjunctionOfConstraintConjunctions & _result,
    bool _accordingPaper,
    Variables & _conflictingVariables,
    const smtrat::EvalDoubleIntervalMap & _solutionSpace )
```

0.14.94.2.33 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.

0.15 Data Structure Documentation

0.15.1 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

0.15.1.1 Constructor & Destructor Documentation

0.15.1.1.1 ~AbstractModuleFactory()

```
virtual smtrat::AbstractModuleFactory::~AbstractModuleFactory ( ) [virtual], [default]
```

0.15.1.2 Member Function Documentation

0.15.1.2.1 create()

```
virtual Module* smtrat::AbstractModuleFactory::create (
    const ModuleInput * _formula,
    Conditionals & _conditionals,
    Manager *const _manager ) [pure virtual]
```

Implemented in smtrat::ModuleFactory< Module >.

0.15.1.2.2 moduleName()

```
virtual std::string smtrat::AbstractModuleFactory::moduleName ( ) const [pure virtual]
```

Implemented in smtrat::ModuleFactory< Module >.

0.15.2 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

0.15.2.1 Detailed Description

Base class for all theories.
A theory represents one or multiple SMT-LIB theories and takes care of converting smtlib constructs into our 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.

0.15.2.2 Constructor & Destructor Documentation

0.15.2.2.1 AbstractTheory() `smtrat::parser::AbstractTheory::AbstractTheory ( ParserState * state ) [inline]`

0.15.2.2.2 ~AbstractTheory() `virtual smtrat::parser::AbstractTheory::~AbstractTheory ( ) [inline], [virtual]`

0.15.2.3 Member Function Documentation

0.15.2.3.1 addConstants()

```
static void smtrat::parser::AbstractTheory::addConstants (
    qi::symbols< char, types::ConstType > & ) [inline], [static]
```

Initialize the global symbol table for constants.

0.15.2.3.2 addSimpleSorts()

```
static void smtrat::parser::AbstractTheory::addSimpleSorts (
    qi::symbols< char, carl::Sort > & ) [inline], [static]
```

Initialize the global symbol table for simple sorts.

0.15.2.3.3 declareVariable()

```
virtual bool smtrat::parser::AbstractTheory::declareVariable (
    const std::string & ,
    const carl::Sort & ,
    types::VariableType & ,
    TheoryError & errors ) [inline], [virtual]
```

Declare a new variable with the given name and the given sort.
Reimplemented in smtrat::parser::UninterpretedTheory, smtrat::parser::CoreTheory, smtrat::parser::BitvectorTheory, and smtrat::parser::ArithmeticTheory.

0.15.2.3.4 expandDistinct()

```
template<typename T , typename Builder >
FormulaT smtrat::parser::AbstractTheory::expandDistinct (
    const std::vector< T > & values,
    const Builder & neqBuilder ) [inline]
```

0.15.2.3.5 functionCall()

```
virtual bool smtrat::parser::AbstractTheory::functionCall (
    const Identifier & ,
    const std::vector< types::TermType > & ,
    types::TermType & ,
    TheoryError & errors ) [inline], [virtual]
```

Resolve another unknown function call.
Reimplemented in smtrat::parser::UninterpretedTheory, smtrat::parser::CoreTheory, smtrat::parser::BitvectorTheory, and smtrat::parser::ArithmeticTheory.

0.15.2.3.6 handleDistinct()

```
virtual bool smtrat::parser::AbstractTheory::handleDistinct (
    const std::vector< types::TermType > & ,
    types::TermType & ,
    TheoryError & errors ) [inline], [virtual]
```

Resolve a distinct operator.
Reimplemented in smtrat::parser::CoreTheory, smtrat::parser::BitvectorTheory, smtrat::parser::ArithmeticTheory, and smtrat::parser::UninterpretedTheory.

0.15.2.3.7 handleITE()

```
virtual bool smtrat::parser::AbstractTheory::handleITE (
    const FormulaT & ,
    const types::TermType & ,
    const types::TermType & ,
    types::TermType & ,
    TheoryError & errors ) [inline], [virtual]
```

Resolve an if-then-else operator.

Reimplemented in smtrat::parser::UninterpretedTheory, smtrat::parser::CoreTheory, smtrat::parser::BitvectorTheory, and smtrat::parser::ArithmeticTheory.

0.15.2.3.8 instantiate()

```
virtual bool smtrat::parser::AbstractTheory::instantiate (
    const types::VariableType & ,
    const types::TermType & ,
    types::TermType & ,
    TheoryError & errors ) [inline], [virtual]
```

Instantiate a variable within a term.
Reimplemented in smtrat::parser::BitvectorTheory, and smtrat::parser::ArithmeticTheory.

0.15.2.3.9 refreshVariable()

```
virtual bool smtrat::parser::AbstractTheory::refreshVariable (
    const types::VariableType & ,
    types::VariableType & ,
    TheoryError & errors ) [inline], [virtual]
```

Reimplemented in smtrat::parser::BitvectorTheory.

0.15.2.3.10 resolveSymbol()

```
virtual bool smtrat::parser::AbstractTheory::resolveSymbol (
    const Identifier & ,
    types::TermType & ,
    TheoryError & errors ) [inline], [virtual]
```

Resolve a symbol that was not declared within the ParserState.
This might be a symbol that actually uses indices, for example bitvector constants.
Reimplemented in smtrat::parser::BitvectorTheory.

0.15.2.4 Field Documentation

0.15.2.4.1 state

```
ParserState* smtrat::parser::AbstractTheory::state
```

0.15.3 smtrat::cad::sample_compare::absvalue Struct Reference

```
#include <SampleComparator.h>
```

0.15.4 smtrat::analyzer::AnalysisSettings Struct Reference

```
#include <settings.h>
```

Data Fields

- bool enabled = false
- std::string analyze_projections = "none"

0.15.4.1 Field Documentation

0.15.4.1.1 analyze_projections

```
std::string smtrat::analyzer::AnalysisSettings::analyze_projections
= "none"
```

0.15.4.1.2 enabled

```
bool smtrat::analyzer::AnalysisSettings::enabled = false
```

0.15.5 smtrat::analyzer::AnalyzerStatistics Struct Reference

`#include <statistics.h>`

Public Member Functions

- template<typename T >
  void add (const std::string &key, const T &value)

0.15.5.1 Member Function Documentation

0.15.5.1.1 add() `template<typename T >`

```
void smtrat::analyzer::AnalyzerStatistics::add (
            const std::string & key,
            const T & value )  [inline]
```

0.15.6 smtrat::covering_ng::formula::node_ds::AND Struct Reference

`#include <FormulaEvaluation.h>`

Data Fields

- std::vector< Node > subformulas

0.15.6.1 Field Documentation

0.15.6.1.1 subformulas `std::vector<Node> smtrat::covering_ng::formula::node_ds::AND::subformulas`

0.15.7 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)

0.15.7.1 Constructor & Destructor Documentation

0.15.7.1.1 AnnotatedBVTerm() [1/4] `smtrat::AnnotatedBVTerm::AnnotatedBVTerm ( ) [inline]`

0.15.7.1.2 AnnotatedBVTerm() [2/4]

```
smtrat::AnnotatedBVTerm::AnnotatedBVTerm (
    const BVAnnotation & _type,
    const carl::BVTerm & _term ) [inline]
```

0.15.7.1.3 AnnotatedBVTerm() [3/4]

```
smtrat::AnnotatedBVTerm::AnnotatedBVTerm (
    std::size_t _width,
    bool _signed,
    Integer _offset = 0 ) [inline]
```

0.15.7.1.4 AnnotatedBVTerm() [4/4]

```
smtrat::AnnotatedBVTerm::AnnotatedBVTerm (
    const BVAnnotation & _type ) [inline]
```

0.15.7.2 Member Function Documentation

0.15.7.2.1 operator()()

```
const carl::BVTerm& smtrat::AnnotatedBVTerm::operator() ( ) const [inline]
```

0.15.7.2.2 term()

```
const carl::BVTerm& smtrat::AnnotatedBVTerm::term ( ) const [inline]
```

0.15.7.2.3 type()

```
const BVAnnotation& smtrat::AnnotatedBVTerm::type ( ) const [inline]
```

0.15.7.3 Friends And Related Function Documentation

0.15.7.3.1 operator<<

```
std::ostream& operator<< (
    std::ostream & _out,
    const AnnotatedBVTerm & _term ) [friend]
```

0.15.8 smtrat::cadcells::representation::approximation::ApxCriteria Class Reference

```
#include <criteria.h>
```

Static Public Member Functions

- static void inform (const Polynomial &p, std::size_t root_index)
- static bool cell (const std::vector< Atom > &constraints)
- static bool level (std::size_t lvl)
- static bool poly (datastructures::Projections &proj, const IR &ir)
- static bool poly_pair (datastructures::Projections &proj, const IR &ir_l, const IR &ir_u)
- static bool side (datastructures::Projections &proj, const IR &ir, datastructures::RootMap::const_iterator start, datastructures::RootMap::const_iterator end)

0.15.8.1 Member Function Documentation

0.15.8.1.1 cell()

```
static bool smtrat::cadcells::representation::approximation::ApxCriteria::cell
(
    const std::vector< Atom > & constraints ) [inline], [static]
```

0.15.8.1.2 inform()

```
static void smtrat::cadcells::representation::approximation::ApxCriteria↩
::inform (
            const Polynomial & p,
            std::size_t root_index )  [inline], [static]
```

0.15.8.1.3 level()

```
static bool smtrat::cadcells::representation::approximation::ApxCriteria↩
::level (
            std::size_t lvl )  [inline], [static]
```

0.15.8.1.4 poly()

```
static bool smtrat::cadcells::representation::approximation::ApxCriteria↩
::poly (
            datastructures::Projections & proj,
            const IR & ir )  [inline], [static]
```

0.15.8.1.5 poly_pair()

```
static bool smtrat::cadcells::representation::approximation::ApxCriteria↩
::poly_pair (
            datastructures::Projections & proj,
            const IR & ir_l,
            const IR & ir_u )  [inline], [static]
```

0.15.8.1.6 side()

```
static bool smtrat::cadcells::representation::approximation::ApxCriteria↩
::side (
            datastructures::Projections & proj,
            const IR & ir,
            datastructures::RootMap::const_iterator start,
            datastructures::RootMap::const_iterator end )  [inline], [static]
```

0.15.9 smtrat::cadcells::representation::approximation::ApxSettings Struct Reference

```
#include <ApproximationSettings.h>
```

Data Fields

- const std::size_t taylor_deg = settings_module().get("apx_taylor_deg", (std::size_t)1)
- const std::size_t maximize_n_iter = settings_module().get("apx_maximize_iter", (std::size_t)5)
- const std::size_t n_sb_iterations = settings_module().get("apx_sb_iter", (std::size_t)2)
- const double root_ratio_lower = settings_module().get("apx_root_ratio_l", (double)0.75)
- const double root_ratio_upper = settings_module().get("apx_root_ratio_u", (double)0.875)
- const std::size_t crit_max_considered = settings_module().get("apx_max_considered", (std::size_t)20)
- const std::size_t crit_max_apx = settings_module().get("apx_max_apx", (std::size_t)50)
- const std::size_t crit_max_constraint_involved = settings_module().get("apx_max_involved", (std::size_t)5)
- const std::size_t crit_max_apx_per_poly = settings_module().get("apx_max_app", (std::size_t)5)
- const std::size_t crit_degree_threshold = settings_module().get("apx_deg_threshold", (std::size_t)5)
- const bool crit_level_enabled = settings_module().get("apx_level_enabled", false)
- const bool crit_considered_count_enabled = settings_module().get("apx_considered_enabled", false)
- const bool crit_apx_count_enabled = settings_module().get("apx_count_enabled", true)
- const bool crit_single_degree_enabled = settings_module().get("apx_single_degree_enabled", true)
- const bool crit_pair_degree_enabled = settings_module().get("apx_pair_degree_enabled", false)
- const bool crit_poly_apx_count_enabled = settings_module().get("apx_poly_count_enabled", false)
- const bool crit_involved_count_enabled = settings_module().get("apx_involved_count_enabled", false)
- const bool crit_side_degree_enabled = settings_module().get("apx_side_degree_enabled", false)

Static Public Attributes

- static constexpr ApxPoly bound = ApxPoly::SIMPLE
- static constexpr ApxPoly between = ApxPoly::SIMPLE
- static constexpr ApxRoot root = ApxRoot::SIMPLE_REPRESENTATION

0.15.9.1 Field Documentation

0.15.9.1.1 between

```
constexpr ApxPoly smtrat::cadcells::representation::approximation::Apx↩
Settings::between = ApxPoly::SIMPLE  [static], [constexpr]
```

0.15.9.1.2 bound

```
constexpr ApxPoly smtrat::cadcells::representation::approximation::Apx↩
Settings::bound = ApxPoly::SIMPLE  [static], [constexpr]
```

0.15.9.1.3 crit_apx_count_enabled

```
const bool smtrat::cadcells::representation::approximation↩
::ApxSettings::crit_apx_count_enabled = settings_module().get("apx_count_enabled", true)
```

0.15.9.1.4 crit_considered_count_enabled

```
const bool smtrat::cadcells::representation::approximation↩
::ApxSettings::crit_considered_count_enabled = settings_module().get("apx_considered_enabled",
false)
```

0.15.9.1.5 crit_degree_threshold

```
const std::size_t smtrat::cadcells::representation::approximation↩
::ApxSettings::crit_degree_threshold = settings_module().get("apx_deg_threshold", (std::size↩
_t)5)
```

0.15.9.1.6 crit_involved_count_enabled

```
const bool smtrat::cadcells::representation::approximation↩
::ApxSettings::crit_involved_count_enabled = settings_module().get("apx_involved_count_↩
enabled", false)
```

0.15.9.1.7 crit_level_enabled

```
const bool smtrat::cadcells::representation::approximation::Apx↩
Settings::crit_level_enabled = settings_module().get("apx_level_enabled", false)
```

0.15.9.1.8 crit_max_apx

```
const std::size_t smtrat::cadcells::representation::approximation::↩
ApxSettings::crit_max_apx = settings_module().get("apx_max_apx", (std::size_t)50)
```

0.15.9.1.9 crit_max_apx_per_poly

```
const std::size_t smtrat::cadcells::representation::approximation↩
::ApxSettings::crit_max_apx_per_poly = settings_module().get("apx_max_app", (std::size_t)5)
```

0.15.9.1.10 crit_max_considered

```
const std::size_t smtrat::cadcells::representation::approximation↩
::ApxSettings::crit_max_considered = settings_module().get("apx_max_considered", (std::size_↩
t)20)
```

0.15.9.1.11 crit_max_constraint_involved

`const std::size_t smtrat::cadcells::representation::approximation::ApxSettings::crit_max_constraint_involved = settings_module().get("apx_max_involved", (std::size_t)5)`

0.15.9.1.12 crit_pair_degree_enabled

`const bool smtrat::cadcells::representation::approximation::ApxSettings::crit_pair_degree_enabled = settings_module().get("apx_pair_degree_enabled", false)`

0.15.9.1.13 crit_poly_apx_count_enabled

`const bool smtrat::cadcells::representation::approximation::ApxSettings::crit_poly_apx_count_enabled = settings_module().get("apx_poly_count_enabled", false)`

0.15.9.1.14 crit_side_degree_enabled

`const bool smtrat::cadcells::representation::approximation::ApxSettings::crit_side_degree_enabled = settings_module().get("apx_side_degree_enabled", false)`

0.15.9.1.15 crit_single_degree_enabled

`const bool smtrat::cadcells::representation::approximation::ApxSettings::crit_single_degree_enabled = settings_module().get("apx_single_degree_enabled", true)`

0.15.9.1.16 maximize_n_iter

`const std::size_t smtrat::cadcells::representation::approximation::ApxSettings::maximize_n_iter = settings_module().get("apx_maximize_iter", (std::size_t)5)`

0.15.9.1.17 n_sb_iterations

`const std::size_t smtrat::cadcells::representation::approximation::ApxSettings::n_sb_iterations = settings_module().get("apx_sb_iter", (std::size_t)2)`

0.15.9.1.18 root

`constexpr ApxRoot smtrat::cadcells::representation::approximation::ApxSettings::root = ApxRoot::SIMPLE_REPRESENTATION [static], [constexpr]`

0.15.9.1.19 root_ratio_lower

`const double smtrat::cadcells::representation::approximation::ApxSettings::root_ratio_lower = settings_module().get("apx_root_ratio_l", (double)0.75)`

0.15.9.1.20 root_ratio_upper

`const double smtrat::cadcells::representation::approximation::ApxSettings::root_ratio_upper = settings_module().get("apx_root_ratio_u", (double)0.875)`

0.15.9.1.21 taylor_deg

`const std::size_t smtrat::cadcells::representation::approximation::ApxSettings::taylor_deg = settings_module().get("apx_taylor_deg", (std::size_t)1)`

0.15.10 benchmax::slurm::ArchiveProperties Struct Reference

All properties needed to archive log files.

`#include <SlurmUtilities.h>`

Data Fields

- std::string filename_archive

  Filename of the archive.
- std::string tmp_dir

  Temporary directory to look for output files.

0.15.10.1 Detailed Description

All properties needed to archive log files.

0.15.10.2 Field Documentation

0.15.10.2.1 filename_archive `std::string benchmax::slurm::ArchiveProperties::filename_archive`
Filename of the archive.

0.15.10.2.2 tmp_dir `std::string benchmax::slurm::ArchiveProperties::tmp_dir`
Temporary directory to look for output files.

0.15.11 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

0.15.11.1 Detailed Description

Implements the theory of arithmetic, including LRA, LIA, NRA and NIA.

0.15.11.2 Constructor & Destructor Documentation

0.15.11.2.1 ArithmeticTheory()

```
smtrat::parser::ArithmeticTheory::ArithmeticTheory (
    ParserState * state )
```

0.15.11.3 Member Function Documentation

0.15.11.3.1 addConstants()

```
static void smtrat::parser::AbstractTheory::addConstants (
    qi::symbols< char, types::ConstType > & ) [inline], [static], [inherited]
```

Initialize the global symbol table for constants.

0.15.11.3.2 addSimpleSorts()

```
void smtrat::parser::ArithmeticTheory::addSimpleSorts (
    qi::symbols< char, carl::Sort > & sorts ) [static]
```

0.15.11.3.3 declareVariable()

```
bool smtrat::parser::ArithmeticTheory::declareVariable (
    const std::string & ,
    const carl::Sort & ,
    types::VariableType & ,
    TheoryError & errors ) [virtual]
```

Declare a new variable with the given name and the given sort.
Reimplemented from smtrat::parser::AbstractTheory.

0.15.11.3.4 expandDistinct()

```
template<typename T , typename Builder >
FormulaT smtrat::parser::AbstractTheory::expandDistinct (
    const std::vector< T > & values,
    const Builder & neqBuilder ) [inline], [inherited]
```

0.15.11.3.5 functionCall()

```
bool smtrat::parser::ArithmeticTheory::functionCall (
    const Identifier & ,
    const std::vector< types::TermType > & ,
    types::TermType & ,
    TheoryError & errors ) [virtual]
```

Resolve another unknown function call.
Reimplemented from smtrat::parser::AbstractTheory.

0.15.11.3.6 handleDistinct()

```
bool smtrat::parser::ArithmeticTheory::handleDistinct (
    const std::vector< types::TermType > & ,
    types::TermType & ,
    TheoryError & errors )  [virtual]
```

Resolve a distinct operator.
Reimplemented from smtrat::parser::AbstractTheory.

0.15.11.3.7 handleITE()

```
bool smtrat::parser::ArithmeticTheory::handleITE (
    const FormulaT & ,
    const types::TermType & ,
    const types::TermType & ,
    types::TermType & ,
    TheoryError & errors )  [virtual]
```

Resolve an if-then-else operator.
Reimplemented from smtrat::parser::AbstractTheory.

0.15.11.3.8 instantiate()

```
bool smtrat::parser::ArithmeticTheory::instantiate (
    const types::VariableType & ,
    const types::TermType & ,
    types::TermType & ,
    TheoryError & errors )  [virtual]
```

Instantiate a variable within a term.
Reimplemented from smtrat::parser::AbstractTheory.

0.15.11.3.9 refreshVariable()

```
virtual bool smtrat::parser::AbstractTheory::refreshVariable (
    const types::VariableType & ,
    types::VariableType & ,
    TheoryError & errors )  [inline], [virtual], [inherited]
```

Reimplemented in smtrat::parser::BitvectorTheory.

0.15.11.3.10 resolveSymbol()

```
virtual bool smtrat::parser::AbstractTheory::resolveSymbol (
    const Identifier & ,
    types::TermType & ,
    TheoryError & errors )  [inline], [virtual], [inherited]
```

Resolve a symbol that was not declared within the ParserState.
This might be a symbol that actually uses indices, for example bitvector constants.
Reimplemented in smtrat::parser::BitvectorTheory.

0.15.11.4 Field Documentation

0.15.11.4.1 mITEs

```
std::map<carl::Variable, std::tuple<FormulaT, Poly, Poly> > smtrat::parser↩
::ArithmeticTheory::mITEs
```

0.15.11.4.2 ops

```
std::map<std::string, arithmetic::OperatorType> smtrat::parser::Arithmetic↩
Theory::ops
```

0.15.11.4.3 state

```
ParserState* smtrat::parser::AbstractTheory::state  [inherited]
```

0.15.12 smtrat::parser::types::ArithmeticTheory Struct Reference

Types of the arithmetic theory.
`#include <TheoryTypes.h>`

Public Types

- typedef mpl::vector< Rational, FixedWidthConstant< Integer > > ConstTypes
- typedef mpl::vector< carl::Variable > VariableTypes
- typedef mpl::vector< carl::Variable, Rational, FixedWidthConstant< Integer >, Poly, carl::MultivariateRoot< Poly > > ExpressionTypes
- typedef mpl::vector< carl::Variable, Rational, FixedWidthConstant< Integer >, Poly, carl::MultivariateRoot< Poly > > TermTypes
- typedef carl::mpl_variant_of< TermTypes >::type TermType

0.15.12.1 Detailed Description

Types of the arithmetic theory.

0.15.12.2 Member Typedef Documentation

0.15.12.2.1 ConstTypes

`typedef mpl::vector<Rational, FixedWidthConstant<Integer> > smtrat::parser::types::Ar`

0.15.12.2.2 ExpressionTypes

`typedef mpl::vector<carl::Variable, Rational, FixedWidthConstant<Integer>, Poly, carl::MultivariateRoot<Poly> > smtrat::parser::types::ArithmeticTheory::ExpressionTypes`

0.15.12.2.3 TermType

`typedef carl::mpl_variant_of<TermTypes>::type smtrat::parser::types::ArithmeticTheory::T`

0.15.12.2.4 TermTypes

`typedef mpl::vector<carl::Variable, Rational, FixedWidthConstant<Integer>, Poly, carl::MultivariateRoot<Poly> > smtrat::parser::types::ArithmeticTheory::TermTypes`

0.15.12.2.5 VariableTypes

`typedef mpl::vector<carl::Variable> smtrat::parser::types::ArithmeticTheory::Variabl`

0.15.13 smtrat::execution::Assertion Struct Reference

`#include <ExecutionState.h>`

Data Fields

- FormulaT formula

0.15.13.1 Field Documentation

0.15.13.1.1 formula

`FormulaT smtrat::execution::Assertion::formula`

0.15.14 smtrat::cad::preprocessor::AssignmentCollector Class Reference

`#include <CADPreprocessor.h>`

Public Types

- using CollectionResult = std::variant< bool, ConstraintT >

  Result of an assignment collection.

Public Member Functions

- AssignmentCollector (Model &model)
- const auto & reasons () const
- auto & reasons ()
- const auto & constraints () const
- auto & constraints ()
- CollectionResult collect (std::map< ConstraintT, ConstraintT > &constraints)

0.15.14.1 Member Typedef Documentation

0.15.14.1.1 CollectionResult `using smtrat::cad::preprocessor::AssignmentCollector::CollectionResult = std::variant<bool,ConstraintT>`

Result of an assignment collection.
true: new assignments were found false: no new assignments were found constraint: found direct conflict

0.15.14.2 Constructor & Destructor Documentation

0.15.14.2.1 AssignmentCollector() `smtrat::cad::preprocessor::AssignmentCollector::AssignmentCollector ( Model & model ) [inline]`

0.15.14.3 Member Function Documentation

0.15.14.3.1 collect() `AssignmentCollector::CollectionResult smtrat::cad::preprocessor::AssignmentCollector::collect ( std::map< ConstraintT, ConstraintT > & constraints )`

0.15.14.3.2 constraints() [1/2] `auto& smtrat::cad::preprocessor::AssignmentCollector::constraints ( ) [inline]`

0.15.14.3.3 constraints() [2/2] `const auto& smtrat::cad::preprocessor::AssignmentCollector::constraints ( ) const [inline]`

0.15.14.3.4 reasons() [1/2] `auto& smtrat::cad::preprocessor::AssignmentCollector::reasons ( ) [inline]`

0.15.14.3.5 reasons() [2/2] `const auto& smtrat::cad::preprocessor::AssignmentCollector::reasons ( ) const [inline]`

0.15.15 smtrat::mcsat::arithmetic::AssignmentFinder Struct Reference

`#include <AssignmentFinder.h>`

Public Member Functions

- std::optional< AssignmentOrConflict > operator() (const mcsat::Bookkeeping &data, carl::Variable var) const
- bool active (const mcsat::Bookkeeping &data, const FormulaT &f) const

0.15.15.1 Member Function Documentation

0.15.15.1.1 active()

```
bool smtrat::mcsat::arithmetic::AssignmentFinder::active (
    const mcsat::Bookkeeping & data,
    const FormulaT & f ) const
```

0.15.15.1.2 operator()()

```
std::optional< AssignmentOrConflict > smtrat::mcsat::arithmetic::↩
AssignmentFinder::operator() (
    const mcsat::Bookkeeping & data,
    carl::Variable var ) const
```

0.15.16 smtrat::mcsat::smtaf::AssignmentFinder< Settings > Struct Template Reference

`#include <AssignmentFinder.h>`

Public Member Functions

- std::optional< AssignmentOrConflict > operator() (const mcsat::Bookkeeping &data, carl::Variable var) const
- bool active (const mcsat::Bookkeeping &, const FormulaT &) const

0.15.16.1 Member Function Documentation

0.15.16.1.1 active()

```
template<class Settings >
bool smtrat::mcsat::smtaf::AssignmentFinder< Settings >::active (
    const mcsat::Bookkeeping & ,
    const FormulaT &  ) const  [inline]
```

0.15.16.1.2 operator()()

```
template<class Settings >
std::optional<AssignmentOrConflict> smtrat::mcsat::smtaf::AssignmentFinder< Settings >↩
::operator() (
    const mcsat::Bookkeeping & data,
    carl::Variable var ) const  [inline]
```

0.15.17 smtrat::mcsat::arithmetic::AssignmentFinder_ctx Class Reference

`#include <AssignmentFinder_ctx.h>`

Public Member Functions

- AssignmentFinder_ctx (const std::vector< carl::Variable > &var_order, carl::Variable var, const Model &model)
- bool addConstraint (const FormulaT &f1)
- bool addMVBound (const FormulaT &f1)
- Covering computeCover ()
- AssignmentOrConflict findAssignment ()

0.15.17.1 Constructor & Destructor Documentation

0.15.17.1.1 AssignmentFinder_ctx()

```
smtrat::mcsat::arithmetic::AssignmentFinder_ctx::AssignmentFinder_ctx (
    const std::vector< carl::Variable > & var_order,
    carl::Variable var,
    const Model & model ) [inline]
```

0.15.17.2 Member Function Documentation

0.15.17.2.1 addConstraint()

```
bool smtrat::mcsat::arithmetic::AssignmentFinder_ctx::addConstraint (
    const FormulaT & f1 ) [inline]
```

0.15.17.2.2 addMVBound()

```
bool smtrat::mcsat::arithmetic::AssignmentFinder_ctx::addMVBound (
    const FormulaT & f1 ) [inline]
```

0.15.17.2.3 computeCover()

```
Covering smtrat::mcsat::arithmetic::AssignmentFinder_ctx::computeCover ( ) [inline]
```

0.15.17.2.4 findAssignment()

```
AssignmentOrConflict smtrat::mcsat::arithmetic::AssignmentFinder_ctx::findAssignment ( ) [inline]
```

0.15.18 smtrat::mcsat::arithmetic::AssignmentFinder_detail Class Reference

```
#include <AssignmentFinder_arithmetic.h>
```

Public Member Functions

- AssignmentFinder_detail (carl::Variable var, const Model &model)
- bool addConstraint (const FormulaT &f)
- bool addMVBound (const FormulaT &f)
- Covering computeCover ()
- AssignmentOrConflict findAssignment ()

0.15.18.1 Constructor & Destructor Documentation

0.15.18.1.1 AssignmentFinder_detail()

```
smtrat::mcsat::arithmetic::AssignmentFinder_detail::↩
AssignmentFinder_detail (
            carl::Variable var,
            const Model & model )  [inline]
```

0.15.18.2 Member Function Documentation

0.15.18.2.1 addConstraint()

```
bool smtrat::mcsat::arithmetic::AssignmentFinder_detail::add↩
Constraint (
            const FormulaT & f )  [inline]
```

0.15.18.2.2 addMVBound()

```
bool smtrat::mcsat::arithmetic::AssignmentFinder_detail::addMVBound
(
            const FormulaT & f )  [inline]
```

0.15.18.2.3 computeCover()

```
Covering smtrat::mcsat::arithmetic::AssignmentFinder_detail::compute↩
Cover ( )  [inline]
```

0.15.18.2.4 findAssignment()

```
AssignmentOrConflict smtrat::mcsat::arithmetic::AssignmentFinder↩
_detail::findAssignment ( )  [inline]
```

0.15.19 smtrat::mcsat::smtaf::AssignmentFinder_SMT Class Reference

```
#include <AssignmentFinder_SMT.h>
```

Public Member Functions

- AssignmentFinder_SMT (VariableRange variables, const Model &model)
- boost::tribool addConstraint (const FormulaT &f)
- boost::tribool addMVBound (const FormulaT &f)
- std::optional< AssignmentOrConflict > findAssignment (const VariablePos excludeVar) const
- std::optional< AssignmentOrConflict > findAssignment () const

0.15.19.1 Constructor & Destructor Documentation

0.15.19.1.1 AssignmentFinder_SMT()

```
smtrat::mcsat::smtaf::AssignmentFinder_SMT::Assignment↩
Finder_SMT (
            VariableRange variables,
            const Model & model )  [inline]
```

0.15.19.2 Member Function Documentation

0.15.19.2.1 addConstraint()

```
boost::tribool smtrat::mcsat::smtaf::AssignmentFinder_SMT::add↩
Constraint (
            const FormulaT & f )
```

0.15.19.2.2 addMVBound()

```
boost::tribool smtrat::mcsat::smtaf::AssignmentFinder_SMT::addMVBound (
            const FormulaT & f )
```

0.15.19.2.3 findAssignment() [1/2]

```
std::optional< AssignmentOrConflict > smtrat::mcsat::smtaf::AssignmentFinder_SMT::findAssignment ( ) const
```

0.15.19.2.4 findAssignment() [2/2]

```
std::optional< AssignmentOrConflict > smtrat::mcsat::smtaf::AssignmentFinder_SMT::findAssignment (
            const VariablePos excludeVar ) const
```

0.15.20 smtrat::cadcells::datastructures::detail::AssignmentProperties Struct Reference

`#include <projections.h>`

Data Fields

- std::map< PolyRef, carl::RealRootsResult< RAN > > real_roots
- std::map< PolyRef, bool > is_zero

0.15.20.1 Field Documentation

0.15.20.1.1 is_zero

```
std::map<PolyRef, bool> smtrat::cadcells::datastructures::detail::AssignmentProperties::is_zero
```

0.15.20.1.2 real_roots

```
std::map<PolyRef, carl::RealRootsResult<RAN> > smtrat::cadcells::datastructures::detail::AssignmentProperties::real_roots
```

0.15.21 smtrat::parser::Attribute Class Reference

Represents an Attribute.

`#include <Attribute.h>`

Public Types

- using AttributeValue = types::AttributeValue

Public Member Functions

- Attribute ()
- Attribute (const std::string &key)
- Attribute (const std::string &key, const AttributeValue &value)
- Attribute (const std::string &key, const boost::optional< AttributeValue > &value)
- bool hasValue () const
- void simplify ()

Data Fields

- std::string key
- AttributeValue value

0.15.21.1 Detailed Description

Represents an Attribute.

0.15.21.2 Member Typedef Documentation

0.15.21.2.1 AttributeValue `using smtrat::parser::Attribute::AttributeValue = types::AttributeValue`

0.15.21.3 Constructor & Destructor Documentation

0.15.21.3.1 Attribute() [1/4] `smtrat::parser::Attribute::Attribute ( ) [inline]`

0.15.21.3.2 Attribute() [2/4]

```
smtrat::parser::Attribute::Attribute (
    const std::string & key ) [inline], [explicit]
```

0.15.21.3.3 Attribute() [3/4]

```
smtrat::parser::Attribute::Attribute (
    const std::string & key,
    const AttributeValue & value ) [inline]
```

0.15.21.3.4 Attribute() [4/4]

```
smtrat::parser::Attribute::Attribute (
    const std::string & key,
    const boost::optional< AttributeValue > & value ) [inline]
```

0.15.21.4 Member Function Documentation

0.15.21.4.1 hasValue() `bool smtrat::parser::Attribute::hasValue ( ) const [inline]`

0.15.21.4.2 simplify() `void smtrat::parser::Attribute::simplify ( ) [inline]`

0.15.21.5 Field Documentation

0.15.21.5.1 key `std::string smtrat::parser::Attribute::key`

0.15.21.5.2 value `AttributeValue smtrat::parser::Attribute::value`

0.15.22 smtrat::parser::AttributeParser Struct Reference

`#include <Attribute.h>`

Public Member Functions

- AttributeParser ()

Data Fields

- KeywordParser keyword
- AttributeValueParser value
- qi::rule< Iterator, Attribute(), Skipper > main

0.15.22.1 Constructor & Destructor Documentation

0.15.22.1.1 AttributeParser() `smtrat::parser::AttributeParser::AttributeParser ( ) [inline]`

0.15.22.2 Field Documentation

0.15.22.2.1 keyword `KeywordParser smtrat::parser::AttributeParser::keyword`

0.15.22.2.2 main `qi::rule<Iterator, Attribute(), Skipper> smtrat::parser::AttributeParser↩ ::main`

0.15.22.2.3 value `AttributeValueParser smtrat::parser::AttributeParser::value`

0.15.23 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

0.15.23.1 Member Typedef Documentation

0.15.23.1.1 Converter `typedef conversion::VariantVariantConverter<types::AttributeValue> smtrat::parser::Attr`

0.15.23.2 Constructor & Destructor Documentation

0.15.23.2.1 AttributeValueParser() `smtrat::parser::AttributeValueParser::AttributeValueParser ( ) [inline]`

0.15.23.3 Field Documentation

0.15.23.3.1 converter `Converter smtrat::parser::AttributeValueParser::converter`

0.15.23.3.2 main `qi::rule<Iterator, types::AttributeValue(), Skipper> smtrat::parser::↩ AttributeValueParser::main`

0.15.23.3.3 sexpression `SExpressionParser smtrat::parser::AttributeValueParser::sexpression`

0.15.23.3.4 specconstant `SpecConstantParser smtrat::parser::AttributeValueParser::specconstant`

0.15.23.3.5 symbol `SymbolParser smtrat::parser::AttributeValueParser::symbol`

0.15.24 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)

0.15.24.1 Member Enumeration Documentation

0.15.24.1.1 AxiomType `enum smtrat::AxiomFactory::AxiomType`

Enumerator

ZERO	
TANGENT_PLANE	
MONOTONICITY	
CONGRUENCE	
ICP	

0.15.24.2 Member Function Documentation

0.15.24.2.1 createFormula()

```
FormulasT smtrat::AxiomFactory::createFormula (
    AxiomType axiomType,
    Model linearizedModel,
    MonomialMap monomialMap ) [static]
```

0.15.25 benchmax::Backend Class Reference

Base class for all backends.
`#include <Backend.h>`

Public Member Functions

- bool suspendable () const
- void process_results (const Jobs &jobs, bool check_finished)
- void addResult (const Tool *tool, const fs::path &file, BenchmarkResult &&result)

  Add a result.
- void run (const Jobs &jobs, bool wait_for_termination)

  Run the list of tools against the list of benchmarks.

Protected Member Functions

- Backend ()
- virtual void startTool (const Tool *)

  Hook for every tool at the beginning.
- virtual void finalize ()

  Hook to allow for asynchronous backends to wait for jobs to terminate.
- virtual void execute (const Tool *, const fs::path &)

  Execute a single pair of tool and benchmark.
- void madeProgress (std::size_t files=1)

  Can be called to give information about the current progress, if available.
- virtual bool collect_results (const Jobs &, bool)
- void sanitize_results (const Jobs &jobs) const
- void write_results (const Jobs &jobs) const

Protected Attributes

- std::size_t mExpectedJobs

  Number of jobs that should be run.
- std::atomic< std::size_t > mFinishedJobs

  Number of jobs that are finished.
- std::atomic< std::size_t > mLastPercent

  Percentage of finished jobs when madeProgress() was last called.

0.15.25.1 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.

0.15.25.2 Constructor & Destructor Documentation

0.15.25.2.1 Backend() `benchmax::Backend::Backend ( ) [inline], [protected]`

0.15.25.3 Member Function Documentation

0.15.25.3.1 addResult()

```
void benchmax::Backend::addResult (
    const Tool * tool,
    const fs::path & file,
    BenchmarkResult && result ) [inline]
```

Add a result.

0.15.25.3.2 collect_results()

```
virtual bool benchmax::Backend::collect_results (
    const Jobs & ,
    bool ) [inline], [protected], [virtual]
```

0.15.25.3.3 execute()

```
virtual void benchmax::Backend::execute (
    const Tool * ,
    const fs::path & ) [inline], [protected], [virtual]
```

Execute a single pair of tool and benchmark.
Reimplemented in benchmax::LocalBackend, and benchmax::CondorBackend.

0.15.25.3.4 finalize()

```
virtual void benchmax::Backend::finalize ( ) [inline], [protected], [virtual]
```

Hook to allow for asynchronous backends to wait for jobs to terminate.

0.15.25.3.5 madeProgress()

```
void benchmax::Backend::madeProgress (
    std::size_t files = 1 ) [inline], [protected]
```

Can be called to give information about the current progress, if available.

0.15.25.3.6 process_results()

```
void benchmax::Backend::process_results (
    const Jobs & jobs,
    bool check_finished ) [inline]
```

0.15.25.3.7 run()

```
void benchmax::Backend::run (
    const Jobs & jobs,
    bool wait_for_termination ) [inline]
```

Run the list of tools against the list of benchmarks.

0.15.25.3.8 sanitize_results()

```
void benchmax::Backend::sanitize_results (
    const Jobs & jobs ) const [inline], [protected]
```

0.15.25.3.9 startTool()

```
virtual void benchmax::Backend::startTool (
    const Tool * ) [inline], [protected], [virtual]
```

Hook for every tool at the beginning.
Can be used to upload the tool to some remote system.

0.15.25.3.10 suspendable()

```
bool benchmax::Backend::suspendable ( ) const [inline]
```

0.15.25.3.11 write_results()

```
void benchmax::Backend::write_results (
    const Jobs & jobs ) const [inline], [protected]
```

0.15.25.4 Field Documentation

0.15.25.4.1 mExpectedJobs `std::size_t benchmax::Backend::mExpectedJobs [protected]`

Number of jobs that should be run.

0.15.25.4.2 mFinishedJobs `std::atomic<std::size_t> benchmax::Backend::mFinishedJobs [protected]`

Number of jobs that are finished.

0.15.25.4.3 mLastPercent `std::atomic<std::size_t> benchmax::Backend::mLastPercent [protected]`

Percentage of finished jobs when madeProgress() was last called.

0.15.26 smtrat::Backend< Settings > Class Template Reference

`#include <Backend.h>`

Public Member Functions

- Backend ()
- void init (std::vector< carl::Variable > &varOrdering)
- std::size_t dimension ()
- std::shared_ptr< cadcells::Polynomial::ContextType > getContext ()
- const carl::Assignment< cadcells::RAN > & getCurrentAssignment ()
- auto & getCoveringInformation ()
- FormulaSetT getInfeasibleSubset ()
- void addConstraint (const ConstraintT &constraint)
- void addConstraint (const size_t level, const std::vector< ConstraintT > &&constraints)
- auto & getUnknownConstraints ()
- auto & getUnknownConstraints (std::size_t &level)
- auto & getKnownConstraints ()
- auto & getKnownConstraints (std::size_t &level)
- void updateAssignment (std::size_t level)
- void resetStoredData (std::size_t level)
- void resetDerivationToConstraintMap ()
- void removeConstraint (const ConstraintT &constraint)
- void setConstraintsKnown (const std::size_t &level)
- void setConstraintsUnknown (const std::size_t &level)
- void processUnknownConstraints (const std::size_t &level, const bool prune_assignment)
- Answer getUnsatCover (const std::size_t level)

0.15.26.1 Constructor & Destructor Documentation

0.15.26.1.1 Backend() `template<typename Settings >`

`smtrat::Backend< Settings >::Backend ( ) [inline]`

0.15.26.2 Member Function Documentation

0.15.26.2.1 addConstraint() [1/2]

```
template<typename Settings >
void smtrat::Backend< Settings >::addConstraint (
            const ConstraintT & constraint )  [inline]
```

0.15.26.2.2 addConstraint() [2/2]

```
template<typename Settings >
void smtrat::Backend< Settings >::addConstraint (
            const size_t level,
            const std::vector< ConstraintT > && constraints )  [inline]
```

0.15.26.2.3 dimension()

```
template<typename Settings >
std::size_t smtrat::Backend< Settings >::dimension ( )  [inline]
```

0.15.26.2.4 getContext()

```
template<typename Settings >
std::shared_ptr<cadcells::Polynomial::ContextType> smtrat::Backend< Settings >::getContext (
)  [inline]
```

0.15.26.2.5 getCoveringInformation()

```
template<typename Settings >
auto& smtrat::Backend< Settings >::getCoveringInformation ( )  [inline]
```

0.15.26.2.6 getCurrentAssignment()

```
template<typename Settings >
const carl::Assignment<cadcells::RAN>& smtrat::Backend< Settings >::getCurrentAssignment ( )
[inline]
```

0.15.26.2.7 getInfeasibleSubset()

```
template<typename Settings >
FormulaSetT smtrat::Backend< Settings >::getInfeasibleSubset ( )  [inline]
```

0.15.26.2.8 getKnownConstraints() [1/2]

```
template<typename Settings >
auto& smtrat::Backend< Settings >::getKnownConstraints ( )  [inline]
```

0.15.26.2.9 getKnownConstraints() [2/2]

```
template<typename Settings >
auto& smtrat::Backend< Settings >::getKnownConstraints (
            std::size_t & level )  [inline]
```

0.15.26.2.10 getUnknownConstraints() [1/2]

```
template<typename Settings >
auto& smtrat::Backend< Settings >::getUnknownConstraints ( )  [inline]
```

0.15.26.2.11 getUnknownConstraints() [2/2]

```
template<typename Settings >
auto& smtrat::Backend< Settings >::getUnknownConstraints (
            std::size_t & level )  [inline]
```

0.15.26.2.12 getUnsatCover()

```
template<typename Settings >
Answer smtrat::Backend< Settings >::getUnsatCover (
            const std::size_t level )  [inline]
```

0.15.26.2.13 init() template<typename Settings >

```
void smtrat::Backend< Settings >::init (
            std::vector< carl::Variable > & varOrdering )  [inline]
```

0.15.26.2.14 processUnknownConstraints() template<typename Settings >

```
void smtrat::Backend< Settings >::processUnknownConstraints (
            const std::size_t & level,
            const bool prune_assignment )  [inline]
```

0.15.26.2.15 removeConstraint() template<typename Settings >

```
void smtrat::Backend< Settings >::removeConstraint (
            const ConstraintT & constraint )  [inline]
```

0.15.26.2.16 resetDerivationToConstraintMap() template<typename Settings >

```
void smtrat::Backend< Settings >::resetDerivationToConstraintMap ( )  [inline]
```

0.15.26.2.17 resetStoredData() template<typename Settings >

```
void smtrat::Backend< Settings >::resetStoredData (
            std::size_t level )  [inline]
```

0.15.26.2.18 setConstraintsKnown() template<typename Settings >

```
void smtrat::Backend< Settings >::setConstraintsKnown (
            const std::size_t & level )  [inline]
```

0.15.26.2.19 setConstraintsUnknown() template<typename Settings >

```
void smtrat::Backend< Settings >::setConstraintsUnknown (
            const std::size_t & level )  [inline]
```

0.15.26.2.20 updateAssignment() template<typename Settings >

```
void smtrat::Backend< Settings >::updateAssignment (
            std::size_t level )  [inline]
```

0.15.27 smtrat::BackendLink Class Reference

```
#include <StrategyGraph.h>
```

Public Member Functions

- BackendLink (std::size_t target, std::size_t priority, const ConditionFunction &cf)
- bool checkCondition (const carl::Condition &c) const
- std::size_t getTarget () const
- std::size_t getPriority () const
- bool operator< (const BackendLink &rhs) const
- BackendLink & priority (std::size_t p)
- template<typename T >
  BackendLink & condition (const T &f)
- BackendLink & id (std::size_t &id)

0.15.27.1 Constructor & Destructor Documentation

0.15.27.1.1 BackendLink()

```
smtrat::BackendLink::BackendLink (
    std::size_t target,
    std::size_t priority,
    const ConditionFunction & cf ) [inline]
```

0.15.27.2 Member Function Documentation

0.15.27.2.1 checkCondition()

```
bool smtrat::BackendLink::checkCondition (
    const carl::Condition & c ) const [inline]
```

0.15.27.2.2 condition()

```
template<typename T >
BackendLink& smtrat::BackendLink::condition (
    const T & f ) [inline]
```

0.15.27.2.3 getPriority()

```
std::size_t smtrat::BackendLink::getPriority ( ) const [inline]
```

0.15.27.2.4 getTarget()

```
std::size_t smtrat::BackendLink::getTarget ( ) const [inline]
```

0.15.27.2.5 id()

```
BackendLink& smtrat::BackendLink::id (
    std::size_t & id ) [inline]
```

0.15.27.2.6 operator<()

```
bool smtrat::BackendLink::operator< (
    const BackendLink & rhs ) const [inline]
```

0.15.27.2.7 priority()

```
BackendLink& smtrat::BackendLink::priority (
    std::size_t p ) [inline]
```

0.15.28 smtrat::BackendSynchronisation Class Reference

```
#include <ThreadPool.h>
```

Public Member Functions

- BackendSynchronisation ()
- void wait ()
- void notify ()

0.15.28.1 Constructor & Destructor Documentation

0.15.28.1.1 BackendSynchronisation()

```
smtrat::BackendSynchronisation::BackendSynchronisation ( ) [inline]
```

0.15.28.2 Member Function Documentation

0.15.28.2.1 notify() `void smtrat::BackendSynchronisation::notify ( ) [inline]`

0.15.28.2.2 wait() `void smtrat::BackendSynchronisation::wait ( ) [inline]`

0.15.29 smtrat::Backtrackable< UnionFind > Struct Template Reference

`#include <UnionFind.h>`

Public Types

- using Value = typename UnionFind::Value
- using Representative = Value
- using Timestamp = std::pair< Value, Value >
- using History = std::vector< std::pair< Timestamp, UnionFind > >

Public Member Functions

- Backtrackable ()
- void init_size (size_t size) noexcept
- auto find (Value const &val) noexcept -> Representative
- void merge (Value const &a, Value const &b) noexcept
- void backtrack (Value const &a, Value const &b) noexcept
- auto version (Timestamp &&ts) const noexcept -> typename History::const_iterator
- constexpr auto origin () const noexcept -> Timestamp
- UnionFind & current () noexcept

Data Fields

- History history

0.15.29.1 Member Typedef Documentation

0.15.29.1.1 History `template<typename UnionFind >`

```
using smtrat::Backtrackable< UnionFind >::History = std::vector< std::pair< Timestamp, Union↩
Find > >
```

0.15.29.1.2 Representative `template<typename UnionFind >`

```
using smtrat::Backtrackable< UnionFind >::Representative = Value
```

0.15.29.1.3 Timestamp `template<typename UnionFind >`

```
using smtrat::Backtrackable< UnionFind >::Timestamp = std::pair<Value, Value>
```

0.15.29.1.4 Value `template<typename UnionFind >`

```
using smtrat::Backtrackable< UnionFind >::Value = typename UnionFind::Value
```

0.15.29.2 Constructor & Destructor Documentation

0.15.29.2.1 Backtrackable()

```
template<typename UnionFind >
smtrat::Backtrackable< UnionFind >::Backtrackable ( ) [inline]
```

0.15.29.3 Member Function Documentation

0.15.29.3.1 backtrack()

```
template<typename UnionFind >
void smtrat::Backtrackable< UnionFind >::backtrack (
            Value const & a,
            Value const & b ) [inline], [noexcept]
```

0.15.29.3.2 current()

```
template<typename UnionFind >
UnionFind& smtrat::Backtrackable< UnionFind >::current ( ) [inline], [noexcept]
```

0.15.29.3.3 find()

```
template<typename UnionFind >
auto smtrat::Backtrackable< UnionFind >::find (
            Value const & val ) -> Representative [inline], [noexcept]
```

0.15.29.3.4 init_size()

```
template<typename UnionFind >
void smtrat::Backtrackable< UnionFind >::init_size (
            size_t size ) [inline], [noexcept]
```

0.15.29.3.5 merge()

```
template<typename UnionFind >
void smtrat::Backtrackable< UnionFind >::merge (
            Value const & a,
            Value const & b ) [inline], [noexcept]
```

0.15.29.3.6 origin()

```
template<typename UnionFind >
constexpr auto smtrat::Backtrackable< UnionFind >::origin ( ) const -> Timestamp [inline],
[constexpr], [noexcept]
```

0.15.29.3.7 version()

```
template<typename UnionFind >
auto smtrat::Backtrackable< UnionFind >::version (
            Timestamp && ts ) const -> typename History::const_iterator [inline], [noexcept]
```

0.15.29.4 Field Documentation

0.15.29.4.1 history

```
template<typename UnionFind >
History smtrat::Backtrackable< UnionFind >::history
```

0.15.30 smtrat::mcsat::onecell::Base Struct Reference

```
#include <onecell.h>
```

Static Public Attributes

- constexpr static bool use_approximation = false

0.15.30.1 Field Documentation

0.15.30.1.1 use_approximation `constexpr static bool smtrat::mcsat::onecell::Base::use_approximation = false [static], [constexpr]`

0.15.31 smtrat::cadcells::datastructures::BaseDerivation< Properties > Class Template Reference

A BaseDerivation has a level and a set of properties of this level, and an underlying derivation representing the lower levels.

`#include <derivation.h>`

Public Member Functions

- BaseDerivation (Projections &projections, DerivationRef< Properties > underlying, size_t level)
- BaseDerivation (const BaseDerivation &other)
- DerivationRef< Properties > & underlying ()
- const DerivationRef< Properties > & underlying () const
- PolyPool & polys ()
- Projections & proj ()
- carl::Variable main_var () const
- size_t level () const
- template<typename P >
  void insert (P property)
- template<typename P >
  bool contains (const P &property) const
- template<typename P >
  const PropertiesTSet< P > & properties () const
- void merge_with (const BaseDerivation< Properties > &other)

Friends

- template<typename P >
  void merge_underlying (std::vector< SampledDerivationRef< P >> &derivations)

0.15.31.1 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)

0.15.31.2 Constructor & Destructor Documentation

0.15.31.2.1 BaseDerivation() [1/2]

```
template<typename Properties >
smtrat::cadcells::datastructures::BaseDerivation< Properties >::BaseDerivation (
            Projections & projections,
```

```
            DerivationRef< Properties > underlying,
            size_t level )  [inline]
```

0.15.31.2.2 BaseDerivation() [2/2]

```
template<typename Properties >
smtrat::cadcells::datastructures::BaseDerivation< Properties >::BaseDerivation (
            const BaseDerivation< Properties > & other )  [inline]
```

0.15.31.3 Member Function Documentation

0.15.31.3.1 contains()

```
template<typename Properties >
template<typename P >
bool smtrat::cadcells::datastructures::BaseDerivation< Properties >::contains (
            const P & property ) const  [inline]
```

0.15.31.3.2 insert()

```
template<typename Properties >
template<typename P >
void smtrat::cadcells::datastructures::BaseDerivation< Properties >::insert (
            P property )  [inline]
```

0.15.31.3.3 level()

```
size_t smtrat::cadcells::datastructures::BaseDerivation< Properties >::level ( ) const  [inline]
```

0.15.31.3.4 main_var()

```
carl::Variable smtrat::cadcells::datastructures::BaseDerivation< Properties >::main_var ( )
const  [inline]
```

0.15.31.3.5 merge_with()

```
void smtrat::cadcells::datastructures::BaseDerivation< Properties >::merge_with (
            const BaseDerivation< Properties > & other )  [inline]
```

0.15.31.3.6 polys()

```
PolyPool& smtrat::cadcells::datastructures::BaseDerivation< Properties >::polys ( )  [inline]
```

0.15.31.3.7 proj()

```
Projections& smtrat::cadcells::datastructures::BaseDerivation< Properties >::proj ( )  [inline]
```

0.15.31.3.8 properties()

```
template<typename Properties >
template<typename P >
const PropertiesTSet<P>& smtrat::cadcells::datastructures::BaseDerivation< Properties >↩
::properties ( ) const  [inline]
```

0.15.31.3.9 underlying() [1/2]

```
template<typename Properties >
DerivationRef<Properties>& smtrat::cadcells::datastructures::BaseDerivation< Properties >↩
::underlying ( )  [inline]
```

0.15.31.3.10 underlying() [2/2]

```
template<typename Properties >
const DerivationRef<Properties>& smtrat::cadcells::datastructures::BaseDerivation< Properties
>::underlying ( ) const [inline]
```

0.15.31.4 Friends And Related Function Documentation

0.15.31.4.1 merge_underlying

```
template<typename Properties >
template<typename P >
void merge_underlying (
            std::vector< SampledDerivationRef< P >> & derivations ) [friend]
```

0.15.32 smtrat::cad::BaseProjection< Settings > Class Template Reference

```
#include <BaseProjection.h>
```

Public Member Functions

- std::size_t dim () const

  Returns the dimension of the projection.
- const auto & vars () const

  Returns the variables used for projection.
- void reset ()

  Resets all datastructures, use the given variables from now on.
- template<typename F >
  void setRemoveCallback (F &&f)

  Sets a callback that is called whenever polynomials are removed.
- carl::Bitset addPolynomial (const Poly &p, std::size_t cid, bool isBound)

  Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.
- virtual carl::Bitset addPolynomial (const UPoly &p, std::size_t cid, bool isBound)=0

  Adds the given polynomial to the projection.
- carl::Bitset addEqConstraint (const Poly &p, std::size_t cid, bool isBound)

  Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.
- virtual carl::Bitset addEqConstraint (const UPoly &p, std::size_t cid, bool isBound)

  Adds the given polynomial of an equational constraint to the projection.
- void removePolynomial (const Poly &p, std::size_t cid, bool isBound)

  Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.
- virtual void removePolynomial (const UPoly &p, std::size_t cid, bool isBound)=0

  Removes the given polynomial from the projection.
- virtual std::size_t size (std::size_t level) const =0
- std::size_t size () const
- virtual bool empty (std::size_t level) const =0
- virtual bool empty ()
- OptionalID getPolyForLifting (std::size_t level, SampleLiftedWith &slw)

  Get a polynomial from this level suited for lifting.
- virtual bool hasPolynomialById (std::size_t level, std::size_t id) const =0
- virtual const UPoly & getPolynomialById (std::size_t level, std::size_t id) const =0

  Retrieves a polynomial from its id.
- virtual void exportAsDot (std::ostream &) const
- virtual Origin getOrigin (std::size_t level, std::size_t id) const

Protected Types

- using Constraints = CADConstraints< Settings::backtracking >

Protected Member Functions

- BaseProjection (const Constraints &c)
- void callRemoveCallback (std::size_t level, const SampleLiftedWith &slw) const
- std::size_t getID (std::size_t level)

  Returns a fresh polynomial id for the given level.
- void freeID (std::size_t level, std::size_t id)

  Frees a currently used polynomial id for the given level.
- carl::Variable var (std::size_t level) const

  Returns the variable that corresponds to the given level, that is the variable eliminated in this level.
- bool canBePurgedByBounds (const UPoly &p) const

  Checks whether a polynomial can safely be ignored due to the bounds.
- bool isPurged (std::size_t level, std::size_t id)

Protected Attributes

- const Constraints & mConstraints
- std::vector< PolynomialLiftingQueue< BaseProjection > > mLiftingQueues

  List of lifting queues that can be used for incremental projection.
- ProjectionInformation mInfo

  Additional info on projection, projection levels and projection polynomials.
- ProjectionOperator mOperator

  The projection operator.
- std::function< void(std::size_t, const SampleLiftedWith &)> mRemoveCallback

  Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.

0.15.32.1 Member Typedef Documentation

0.15.32.1.1 Constraints

```
template<typename Settings >
using smtrat::cad::BaseProjection< Settings >::Constraints = CADConstraints<Settings::backtracking>
[protected]
```

0.15.32.2 Constructor & Destructor Documentation

0.15.32.2.1 BaseProjection()

```
template<typename Settings >
smtrat::cad::BaseProjection< Settings >::BaseProjection (
            const Constraints & c )  [inline], [protected]
```

0.15.32.3 Member Function Documentation

0.15.32.3.1 addEqConstraint() [1/2] template<typename Settings >

```
carl::Bitset smtrat::cad::BaseProjection< Settings >::addEqConstraint (
            const Poly & p,
            std::size_t cid,
            bool isBound ) [inline]
```

Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.

0.15.32.3.2 addEqConstraint() [2/2] template<typename Settings >

```
virtual carl::Bitset smtrat::cad::BaseProjection< Settings >::addEqConstraint (
            const UPoly & p,
            std::size_t cid,
            bool isBound ) [inline], [virtual]
```

Adds the given polynomial of an equational constraint to the projection.
Reimplemented in smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >.

0.15.32.3.3 addPolynomial() [1/2] template<typename Settings >

```
carl::Bitset smtrat::cad::BaseProjection< Settings >::addPolynomial (
            const Poly & p,
            std::size_t cid,
            bool isBound ) [inline]
```

Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.

0.15.32.3.4 addPolynomial() [2/2] template<typename Settings >

```
virtual carl::Bitset smtrat::cad::BaseProjection< Settings >::addPolynomial (
            const UPoly & p,
            std::size_t cid,
            bool isBound ) [pure virtual]
```

Adds the given polynomial to the projection.
Implemented in smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >, smtrat::cad::Projection< In
smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Increme
smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, and smtrat::cad::Projection< Incrementality::SIMPLE

0.15.32.3.5 callRemoveCallback() template<typename Settings >

```
void smtrat::cad::BaseProjection< Settings >::callRemoveCallback (
            std::size_t level,
            const SampleLiftedWith & slw ) const [inline], [protected]
```

0.15.32.3.6 canBePurgedByBounds() template<typename Settings >

```
bool smtrat::cad::BaseProjection< Settings >::canBePurgedByBounds (
            const UPoly & p ) const [inline], [protected]
```

Checks whether a polynomial can safely be ignored due to the bounds.

0.15.32.3.7 dim() template<typename Settings >

```
std::size_t smtrat::cad::BaseProjection< Settings >::dim ( ) const [inline]
```

Returns the dimension of the projection.

0.15.32.3.8 empty() [1/2] template<typename Settings >

```
virtual bool smtrat::cad::BaseProjection< Settings >::empty ( ) [inline], [virtual]
```

0.15.32.3.9 empty() [2/2] template<typename Settings >

```
virtual bool smtrat::cad::BaseProjection< Settings >::empty (
            std::size_t level ) const [pure virtual]
```

Implemented in smtrat::qe::cad::Projection< Settings >, smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED
smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::ModelBasedProjection< Increme
smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, and smtrat::cad::Projection< Incrementality::FULL, Backtracking::HID

0.15.32.3.10 exportAsDot() template<typename Settings >

```
virtual void smtrat::cad::BaseProjection< Settings >::exportAsDot (
            std::ostream & ) const [inline], [virtual]
```

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, and smtrat::cad::Projection< Incrementality::FULL,

0.15.32.3.11 freeID() template<typename Settings >

```
void smtrat::cad::BaseProjection< Settings >::freeID (
            std::size_t level,
            std::size_t id ) [inline], [protected]
```

Frees a currently used polynomial id for the given level.

0.15.32.3.12 getID() template<typename Settings >

```
std::size_t smtrat::cad::BaseProjection< Settings >::getID (
            std::size_t level ) [inline], [protected]
```

Returns a fresh polynomial id for the given level.

0.15.32.3.13 getOrigin() template<typename Settings >

```
virtual Origin smtrat::cad::BaseProjection< Settings >::getOrigin (
            std::size_t level,
            std::size_t id ) const [inline], [virtual]
```

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, BT, Settings >.

0.15.32.3.14 getPolyForLifting() template<typename Settings >

```
OptionalID smtrat::cad::BaseProjection< Settings >::getPolyForLifting (
            std::size_t level,
            SampleLiftedWith & slw ) [inline]
```

Get a polynomial from this level suited for lifting.

0.15.32.3.15 getPolynomialById() template<typename Settings >

```
virtual const UPoly& smtrat::cad::BaseProjection< Settings >::getPolynomialById (
            std::size_t level,
            std::size_t id ) const [pure virtual]
```

Retrieves a polynomial from its id.
Implemented in smtrat::qe::cad::Projection< Settings >, smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED
smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::ModelBasedProjection< Increme
smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, and smtrat::cad::Projection< Incrementality::FULL, Backtracking::HID

0.15.32.3.16 hasPolynomialById() template<typename Settings >

```
virtual bool smtrat::cad::BaseProjection< Settings >::hasPolynomialById (
            std::size_t level,
            std::size_t id ) const [pure virtual]
```

Implemented in smtrat::qe::cad::Projection< Settings >, smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED
smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::ModelBasedProjection< Increme
smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, and smtrat::cad::Projection< Incrementality::FULL, Backtracking::HID

0.15.32.3.17 isPurged()

```
template<typename Settings >
bool smtrat::cad::BaseProjection< Settings >::isPurged (
            std::size_t level,
            std::size_t id )  [inline], [protected]
```

0.15.32.3.18 removePolynomial() [1/2]

```
template<typename Settings >
void smtrat::cad::BaseProjection< Settings >::removePolynomial (
            const Poly & p,
            std::size_t cid,
            bool isBound )  [inline]
```

Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.

0.15.32.3.19 removePolynomial() [2/2]

```
template<typename Settings >
virtual void smtrat::cad::BaseProjection< Settings >::removePolynomial (
            const UPoly & p,
            std::size_t cid,
            bool isBound )  [pure virtual]
```

Removes the given polynomial from the projection.
Implemented in smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::ModelBasedProje
smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, S
smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >, and smtrat::cad::Projection< Incrementality::NONE, Backtracking::

0.15.32.3.20 reset()

```
template<typename Settings >
void smtrat::cad::BaseProjection< Settings >::reset ( )  [inline]
```

Resets all datastructures, use the given variables from now on.

0.15.32.3.21 setRemoveCallback()

```
template<typename Settings >
template<typename F >
void smtrat::cad::BaseProjection< Settings >::setRemoveCallback (
            F && f )  [inline]
```

Sets a callback that is called whenever polynomials are removed.

0.15.32.3.22 size() [1/2]

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::size ( ) const  [inline]
```

0.15.32.3.23 size() [2/2]

```
template<typename Settings >
virtual std::size_t smtrat::cad::BaseProjection< Settings >::size (
            std::size_t level ) const  [pure virtual]
```

Implemented in smtrat::qe::cad::Projection< Settings >, smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED
smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::ModelBasedProjection< Increme
smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, and smtrat::cad::Projection< Incrementality::FULL, Backtracking::HID

0.15.32.3.24 var() template<typename Settings >

```
carl::Variable smtrat::cad::BaseProjection< Settings >::var (
            std::size_t level ) const [inline], [protected]
```

Returns the variable that corresponds to the given level, that is the variable eliminated in this level.

0.15.32.3.25 vars() template<typename Settings >

```
const auto& smtrat::cad::BaseProjection< Settings >::vars ( ) const [inline]
```

Returns the variables used for projection.

0.15.32.4 Field Documentation

0.15.32.4.1 mConstraints template<typename Settings >

```
const Constraints& smtrat::cad::BaseProjection< Settings >::mConstraints [protected]
```

0.15.32.4.2 mInfo template<typename Settings >

```
ProjectionInformation smtrat::cad::BaseProjection< Settings >::mInfo [protected]
```

Additional info on projection, projection levels and projection polynomials.

0.15.32.4.3 mLiftingQueues template<typename Settings >

```
std::vector<PolynomialLiftingQueue<BaseProjection> > smtrat::cad::BaseProjection< Settings
>::mLiftingQueues [protected]
```

List of lifting queues that can be used for incremental projection.

0.15.32.4.4 mOperator template<typename Settings >

```
ProjectionOperator smtrat::cad::BaseProjection< Settings >::mOperator [protected]
```

The projection operator.

0.15.32.4.5 mRemoveCallback template<typename Settings >

```
std::function<void(std::size_t, const SampleLiftedWith&)> smtrat::cad::BaseProjection< Settings
>::mRemoveCallback [protected]
```

Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.

0.15.33 smtrat::cad::BaseSettings Struct Reference

```
#include <Settings.h>
```

Static Public Attributes

- static constexpr Incrementality incrementality = Incrementality::NONE
- static constexpr Backtracking backtracking = Backtracking::ORDERED
- static constexpr ProjectionType projectionOperator = cad::ProjectionType::McCallum
- static constexpr CoreHeuristic coreHeuristic = cad::CoreHeuristic::PreferProjection
- static constexpr MISHeuristic misHeuristic = cad::MISHeuristic::GREEDY
- static constexpr std::size_t trivialSampleRadius = 1
- static constexpr bool simplifyProjectionByBounds = true
- static constexpr ProjectionCompareStrategy projectionComparator = cad::ProjectionCompareStrategy::Default
- static constexpr SampleCompareStrategy sampleComparator = cad::SampleCompareStrategy::Default
- static constexpr FullSampleCompareStrategy fullSampleComparator = cad::FullSampleCompareStrategy::Default

0.15.33.1 Field Documentation

0.15.33.1.1 backtracking `constexpr Backtracking smtrat::cad::BaseSettings::backtracking = Backtracking::ORDERE` `[static], [constexpr]`

0.15.33.1.2 coreHeuristic `constexpr CoreHeuristic smtrat::cad::BaseSettings::coreHeuristic = cad::CoreHeuristic::PreferProjection [static], [constexpr]`

0.15.33.1.3 fullSampleComparator `constexpr FullSampleCompareStrategy smtrat::cad::BaseSettings↩ ::fullSampleComparator = cad::FullSampleCompareStrategy::Default [static], [constexpr]`

0.15.33.1.4 incrementality `constexpr Incrementality smtrat::cad::BaseSettings::incrementality = Incrementality::NONE [static], [constexpr]`

0.15.33.1.5 misHeuristic `constexpr MISHeuristic smtrat::cad::BaseSettings::misHeuristic = cad::MISHeuristic::GR` `[static], [constexpr]`

0.15.33.1.6 projectionComparator `constexpr ProjectionCompareStrategy smtrat::cad::BaseSettings↩ ::projectionComparator = cad::ProjectionCompareStrategy::Default [static], [constexpr]`

0.15.33.1.7 projectionOperator `constexpr ProjectionType smtrat::cad::BaseSettings::projection↩ Operator = cad::ProjectionType::McCallum [static], [constexpr]`

0.15.33.1.8 sampleComparator `constexpr SampleCompareStrategy smtrat::cad::BaseSettings↩ ::sampleComparator = cad::SampleCompareStrategy::Default [static], [constexpr]`

0.15.33.1.9 simplifyProjectionByBounds `constexpr bool smtrat::cad::BaseSettings::simplify↩ ProjectionByBounds = true [static], [constexpr]`

0.15.33.1.10 trivialSampleRadius `constexpr std::size_t smtrat::cad::BaseSettings::trivial↩ SampleRadius = 1 [static], [constexpr]`

0.15.34 smtrat::expression::simplifier::BaseSimplifier Struct Reference

`#include <BaseSimplifier.h>`

Public Member Functions

- const ExpressionContent ∗ operator() (const carl::Variable &expr) const
- const ExpressionContent ∗ operator() (const ITEExpression &expr) const
- const ExpressionContent ∗ operator() (const QuantifierExpression &expr) const
- const ExpressionContent ∗ operator() (const UnaryExpression &expr) const
- const ExpressionContent ∗ operator() (const BinaryExpression &expr) const
- const ExpressionContent ∗ operator() (const NaryExpression &expr) const
- const ExpressionContent ∗ operator() (const ExpressionContent ∗_ec) const

Protected Member Functions

- virtual const ExpressionContent * simplify (const carl::Variable &) const
- virtual const ExpressionContent * simplify (const ITEExpression &) const
- virtual const ExpressionContent * simplify (const QuantifierExpression &) const
- virtual const ExpressionContent * simplify (const UnaryExpression &) const
- virtual const ExpressionContent * simplify (const BinaryExpression &) const
- virtual const ExpressionContent * simplify (const NaryExpression &) const

0.15.34.1 Member Function Documentation

0.15.34.1.1 operator()() [1/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const BinaryExpression & expr ) const [inline]
```

0.15.34.1.2 operator()() [2/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const carl::Variable & expr ) const [inline]
```

0.15.34.1.3 operator()() [3/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const ExpressionContent * _ec ) const [inline]
```

0.15.34.1.4 operator()() [4/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const ITEExpression & expr ) const [inline]
```

0.15.34.1.5 operator()() [5/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const NaryExpression & expr ) const [inline]
```

0.15.34.1.6 operator()() [6/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const QuantifierExpression & expr ) const [inline]
```

0.15.34.1.7 operator()() [7/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const UnaryExpression & expr ) const [inline]
```

0.15.34.1.8 simplify() [1/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const BinaryExpression & ) const [inline], [protected], [virtual]
```

0.15.34.1.9 simplify() [2/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const carl::Variable & ) const [inline], [protected], [virtual]
```

0.15.34.1.10 simplify() [3/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::↩
BaseSimplifier::simplify (
            const ITEExpression &  ) const  [inline], [protected], [virtual]
```

0.15.34.1.11 simplify() [4/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::↩
BaseSimplifier::simplify (
            const NaryExpression &  ) const  [inline], [protected], [virtual]
```

Reimplemented in smtrat::expression::simplifier::SingletonSimplifier, smtrat::expression::simplifier::MergeSimplifier, and smtrat::expression::simplifier::DuplicateSimplifier.

0.15.34.1.12 simplify() [5/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::↩
BaseSimplifier::simplify (
            const QuantifierExpression &  ) const  [inline], [protected], [virtual]
```

0.15.34.1.13 simplify() [6/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::↩
BaseSimplifier::simplify (
            const UnaryExpression &  ) const  [inline], [protected], [virtual]
```

Reimplemented in smtrat::expression::simplifier::NegationSimplifier.

0.15.35 smtrat::cad::Origin::BaseType Struct Reference

```
#include <Origin.h>
```

Public Member Functions

- BaseType (std::size_t level, std::size_t id)
- BaseType (std::size_t lvl, std::size_t id1, std::size_t id2)
- bool active () const
- void activate (const carl::Bitset &ids)
- void deactivate (const carl::Bitset &ids)
- bool operator== (const BaseType &bt) const
- bool operator< (const BaseType &bt) const

Data Fields

- std::size_t level
- std::size_t first
- std::size_t second
- bool first_active = true
- bool second_active = true
- bool ec_active = true

Friends

- std::ostream & operator<< (std::ostream &os, const BaseType &bt)

0.15.35.1 Constructor & Destructor Documentation

0.15.35.1.1 BaseType() [1/2]

```
smtrat::cad::Origin::BaseType::BaseType (
            std::size_t level,
            std::size_t id )  [inline], [explicit]
```

0.15.35.1.2 BaseType() [2/2]

```
smtrat::cad::Origin::BaseType::BaseType (
    std::size_t lvl,
    std::size_t id1,
    std::size_t id2 ) [inline]
```

0.15.35.2 Member Function Documentation

0.15.35.2.1 activate()

```
void smtrat::cad::Origin::BaseType::activate (
    const carl::Bitset & ids ) [inline]
```

0.15.35.2.2 active()

```
bool smtrat::cad::Origin::BaseType::active ( ) const [inline]
```

0.15.35.2.3 deactivate()

```
void smtrat::cad::Origin::BaseType::deactivate (
    const carl::Bitset & ids ) [inline]
```

0.15.35.2.4 operator<()

```
bool smtrat::cad::Origin::BaseType::operator< (
    const BaseType & bt ) const [inline]
```

0.15.35.2.5 operator==()

```
bool smtrat::cad::Origin::BaseType::operator== (
    const BaseType & bt ) const [inline]
```

0.15.35.3 Friends And Related Function Documentation

0.15.35.3.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const BaseType & bt ) [friend]
```

0.15.35.4 Field Documentation

0.15.35.4.1 ec_active

```
bool smtrat::cad::Origin::BaseType::ec_active = true
```

0.15.35.4.2 first

```
std::size_t smtrat::cad::Origin::BaseType::first
```

0.15.35.4.3 first_active

```
bool smtrat::cad::Origin::BaseType::first_active = true
```

0.15.35.4.4 level

```
std::size_t smtrat::cad::Origin::BaseType::level
```

0.15.35.4.5 second

```
std::size_t smtrat::cad::Origin::BaseType::second
```

0.15.35.4.6 second_active

```
bool smtrat::cad::Origin::BaseType::second_active = true
```

0.15.36 smtrat::mcsat::onecell::BCApproximationSettings Struct Reference

`#include <onecell.h>`

Static Public Attributes

- constexpr static bool use_approximation = true
- constexpr static auto cell_apx_heuristic = cadcells::representation::BIGGEST_CELL_APPROXIMATION
- constexpr static auto cell_heuristic = cadcells::representation::BIGGEST_CELL
- constexpr static auto covering_heuristic = cadcells::representation::BIGGEST_CELL_COVERING
- constexpr static auto op = cadcells::operators::op::mccallum

0.15.36.1 Field Documentation

0.15.36.1.1 cell_apx_heuristic

`constexpr static auto smtrat::mcsat::onecell::BCApproximation↩ Settings::cell_apx_heuristic = cadcells::representation::BIGGEST_CELL_APPROXIMATION [static], [constexpr]`

0.15.36.1.2 cell_heuristic

`constexpr static auto smtrat::mcsat::onecell::BCSettings::cell_↩ heuristic = cadcells::representation::BIGGEST_CELL [static], [constexpr], [inherited]`

0.15.36.1.3 covering_heuristic

`constexpr static auto smtrat::mcsat::onecell::BCSettings::covering↩ _heuristic = cadcells::representation::BIGGEST_CELL_COVERING [static], [constexpr], [inherited]`

0.15.36.1.4 op

`constexpr static auto smtrat::mcsat::onecell::BCSettings::op = cadcells::operators↩ ::op::mccallum [static], [constexpr], [inherited]`

0.15.36.1.5 use_approximation

`constexpr static bool smtrat::mcsat::onecell::BCApproximation↩ Settings::use_approximation = true [static], [constexpr]`

0.15.37 smtrat::mcsat::onecell::BCFilteredAllSelectiveSettings Struct Reference

`#include <onecell.h>`

Static Public Attributes

- constexpr static auto op = cadcells::operators::op::mccallum_filtered_all_selective
- constexpr static auto cell_heuristic = cadcells::representation::BIGGEST_CELL_EW
- constexpr static auto covering_heuristic = cadcells::representation::BIGGEST_CELL_COVERING_EW
- constexpr static bool use_approximation = false

0.15.37.1 Field Documentation

0.15.37.1.1 cell_heuristic

`constexpr static auto smtrat::mcsat::onecell::BCFilteredSettings↩ ::cell_heuristic = cadcells::representation::BIGGEST_CELL_EW [static], [constexpr], [inherited]`

0.15.37.1.2 covering_heuristic

`constexpr static auto smtrat::mcsat::onecell::BCFilteredSettings↩ ::covering_heuristic = cadcells::representation::BIGGEST_CELL_COVERING_EW [static], [constexpr], [inherited]`

0.15.37.1.3 op

`constexpr static auto smtrat::mcsat::onecell::BCFilteredAllSelectiveSettings↩ ::op = cadcells::operators::op::mccallum_filtered_all_selective [static], [constexpr]`

0.15.37.1.4 use_approximation

`constexpr static bool smtrat::mcsat::onecell::Base::use_approximation = false [static], [constexpr], [inherited]`

0.15.38 smtrat::mcsat::onecell::BCFilteredAllSettings Struct Reference

`#include <onecell.h>`

Static Public Attributes

- constexpr static auto op = cadcells::operators::op::mccallum_filtered_all
- constexpr static auto cell_heuristic = cadcells::representation::BIGGEST_CELL_EW
- constexpr static auto covering_heuristic = cadcells::representation::BIGGEST_CELL_COVERING_EW
- constexpr static bool use_approximation = false

0.15.38.1 Field Documentation

0.15.38.1.1 cell_heuristic

`constexpr static auto smtrat::mcsat::onecell::BCFilteredSettings↩ ::cell_heuristic = cadcells::representation::BIGGEST_CELL_EW [static], [constexpr], [inherited]`

0.15.38.1.2 covering_heuristic

`constexpr static auto smtrat::mcsat::onecell::BCFilteredSettings↩ ::covering_heuristic = cadcells::representation::BIGGEST_CELL_COVERING_EW [static], [constexpr], [inherited]`

0.15.38.1.3 op

`constexpr static auto smtrat::mcsat::onecell::BCFilteredAllSettings::op = cadcells↩ ::operators::op::mccallum_filtered_all [static], [constexpr]`

0.15.38.1.4 use_approximation

`constexpr static bool smtrat::mcsat::onecell::Base::use_approximation = false [static], [constexpr], [inherited]`

0.15.39 smtrat::mcsat::onecell::BCFilteredBoundsSettings Struct Reference

`#include <onecell.h>`

Static Public Attributes

- constexpr static auto op = cadcells::operators::op::mccallum_filtered_bounds
- constexpr static auto cell_heuristic = cadcells::representation::BIGGEST_CELL_EW
- constexpr static auto covering_heuristic = cadcells::representation::BIGGEST_CELL_COVERING_EW
- constexpr static bool use_approximation = false

0.15.39.1 Field Documentation

0.15.39.1.1 cell_heuristic

`constexpr static auto smtrat::mcsat::onecell::BCFilteredSettings↩ ::cell_heuristic = cadcells::representation::BIGGEST_CELL_EW [static], [constexpr], [inherited]`

0.15.39.1.2 covering_heuristic

`constexpr static auto smtrat::mcsat::onecell::BCFilteredSettings↩ ::covering_heuristic = cadcells::representation::BIGGEST_CELL_COVERING_EW [static], [constexpr], [inherited]`

0.15.39.1.3 op

`constexpr static auto smtrat::mcsat::onecell::BCFilteredBoundsSettings::op = cadcells::operators::op::mccallum_filtered_bounds [static], [constexpr]`

0.15.39.1.4 use_approximation

`constexpr static bool smtrat::mcsat::onecell::Base::use_approximation = false [static], [constexpr], [inherited]`

0.15.40 smtrat::mcsat::onecell::BCFilteredSamplesSettings Struct Reference

`#include <onecell.h>`

Static Public Attributes

- constexpr static auto op = cadcells::operators::op::mccallum_filtered_samples
- constexpr static auto cell_heuristic = cadcells::representation::BIGGEST_CELL_EW
- constexpr static auto covering_heuristic = cadcells::representation::BIGGEST_CELL_COVERING_EW
- constexpr static bool use_approximation = false

0.15.40.1 Field Documentation

0.15.40.1.1 cell_heuristic

`constexpr static auto smtrat::mcsat::onecell::BCFilteredSettings↩ ::cell_heuristic = cadcells::representation::BIGGEST_CELL_EW [static], [constexpr], [inherited]`

0.15.40.1.2 covering_heuristic

`constexpr static auto smtrat::mcsat::onecell::BCFilteredSettings↩ ::covering_heuristic = cadcells::representation::BIGGEST_CELL_COVERING_EW [static], [constexpr], [inherited]`

0.15.40.1.3 op

`constexpr static auto smtrat::mcsat::onecell::BCFilteredSamplesSettings::op = cadcells::operators::op::mccallum_filtered_samples [static], [constexpr]`

0.15.40.1.4 use_approximation

`constexpr static bool smtrat::mcsat::onecell::Base::use_approximation = false [static], [constexpr], [inherited]`

0.15.41 smtrat::mcsat::onecell::BCFilteredSettings Struct Reference

`#include <onecell.h>`

Static Public Attributes

- constexpr static auto cell_heuristic = cadcells::representation::BIGGEST_CELL_EW
- constexpr static auto covering_heuristic = cadcells::representation::BIGGEST_CELL_COVERING_EW
- constexpr static auto op = cadcells::operators::op::mccallum_filtered
- constexpr static bool use_approximation = false

0.15.41.1 Field Documentation

0.15.41.1.1 cell_heuristic

`constexpr static auto smtrat::mcsat::onecell::BCFilteredSettings::cell_heuristic = cadcells::representation::BIGGEST_CELL_EW [static], [constexpr]`

0.15.41.1.2 covering_heuristic

`constexpr static auto smtrat::mcsat::onecell::BCFilteredSettings::covering_heuristic = cadcells::representation::BIGGEST_CELL_COVERING_EW [static], [constexpr]`

0.15.41.1.3 op

`constexpr static auto smtrat::mcsat::onecell::BCFilteredSettings::op = cadcells::operators::op::mccallum_filtered [static], [constexpr]`

0.15.41.1.4 use_approximation

`constexpr static bool smtrat::mcsat::onecell::Base::use_approximation = false [static], [constexpr], [inherited]`

0.15.42 smtrat::mcsat::onecell::BCSettings Struct Reference

`#include <onecell.h>`

Static Public Attributes

- constexpr static auto cell_heuristic = cadcells::representation::BIGGEST_CELL
- constexpr static auto covering_heuristic = cadcells::representation::BIGGEST_CELL_COVERING
- constexpr static auto op = cadcells::operators::op::mccallum
- constexpr static bool use_approximation = false

0.15.42.1 Field Documentation

0.15.42.1.1 cell_heuristic

`constexpr static auto smtrat::mcsat::onecell::BCSettings::cell_heuristic = cadcells::representation::BIGGEST_CELL [static], [constexpr]`

0.15.42.1.2 covering_heuristic

`constexpr static auto smtrat::mcsat::onecell::BCSettings::covering_heuristic = cadcells::representation::BIGGEST_CELL_COVERING [static], [constexpr]`

0.15.42.1.3 op

`constexpr static auto smtrat::mcsat::onecell::BCSettings::op = cadcells::operators::op::mccallum [static], [constexpr]`

0.15.42.1.4 use_approximation

`constexpr static bool smtrat::mcsat::onecell::Base::use_approximation = false [static], [constexpr], [inherited]`

0.15.43 smtrat::BEModule< Settings > Class Template Reference

`#include <BEModule.h>`

Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- BEModule (const ModuleInput ∗_formula, Conditionals &_conditionals, Manager ∗_manager=NULL)
- ∼BEModule ()
- Answer checkCore ()

  Checks the received formula for consistency.
- bool isPreprocessor () const
- bool appliedPreprocessing () const
- bool add (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- void remove (ModuleInput::const_iterator _subformula)

  Removes everything related to the given sub-formula of the received formula.
- Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)

  Checks the received formula for consistency.
- Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- std::pair< bool, FormulaT > getReceivedFormulaSimplified ()
- void updateModel () const

  Updates the current assignment into the model.
- bool inform (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void deinform (const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual void init ()

  Informs all backends about the so far encountered constraints, which have not yet been communicated.
- virtual void updateAllModels ()

  Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
- virtual std::list< std::vector< carl::Variable > > getModelEqualities () const

  Partition the variables from the current model into equivalence classes according to their assigned value.
- unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const
- virtual unsigned currentlySatisfied (const FormulaT &) const
- bool receivedVariable (carl::Variable::Arg _var) const
- Answer solverState () const
- std::size_t id () const
- void setId (std::size_t _id)

  Sets this modules unique ID to identify itself.
- thread_priority threadPriority () const
- void setThreadPriority (thread_priority _threadPriority)

  Sets the priority of this module to get a thread for running its check procedure.
- const ModuleInput ∗ pReceivedFormula () const
- const ModuleInput & rReceivedFormula () const
- const ModuleInput ∗ pPassedFormula () const
- const ModuleInput & rPassedFormula () const
- const Model & model () const
- const std::vector< Model > & allModels () const
- const std::vector< FormulaSetT > & infeasibleSubsets () const
- const std::vector< Module ∗ > & usedBackends () const
- const carl::FastSet< FormulaT > & constraintsToInform () const
- const carl::FastSet< FormulaT > & informedConstraints () const
- void addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))

Stores a lemma being a valid formula.

- bool hasLemmas ()

  Checks whether this module or any of its backends provides any lemmas.
- void clearLemmas ()

  Deletes all yet found lemmas.
- const std::vector< Lemma > & lemmas () const
- ModuleInput::const_iterator firstUncheckedReceivedSubformula () const
- ModuleInput::const_iterator firstSubformulaToPass () const
- void receivedFormulaChecked ()

  Notifies that the received formulas has been checked.
- const 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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.
- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)
- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

Copies the infeasible subsets of the passed formula.

- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

Stores the assignment of the current satisfiable result, if existent.

- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.43.1 Member Typedef Documentation

0.15.43.1.1 SettingsType

```
template<typename Settings >
typedef Settings smtrat::BEModule< Settings >::SettingsType
```

0.15.43.2 Member Enumeration Documentation

0.15.43.2.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

0.15.43.3 Constructor & Destructor Documentation

0.15.43.3.1 BEModule() `template<typename Settings >`

```
smtrat::BEModule< Settings >::BEModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = NULL )
```

0.15.43.3.2 ~BEModule() `template<typename Settings >`

```
smtrat::BEModule< Settings >::~BEModule ( )
```

0.15.43.4 Member Function Documentation

0.15.43.4.1 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.

0.15.43.4.2 addConstraintToInform()

```
void smtrat::Module::addConstraintToInform (
            const FormulaT & _constraint )  [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

0.15.43.4.3 addCore()

```
virtual bool smtrat::Module::addCore (
            [[maybe_unused] ] 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.

0.15.43.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.43.4.5 addLemma()

```
void smtrat::Module::addLemma (
    const FormulaT & _lemma,
    const LemmaType & _lt = LemmaType::NORMAL,
    const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline],
[inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.43.4.6 addOrigin()

```
void smtrat::Module::addOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.43.4.7 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.43.4.8 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula )  [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.43.4.9 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula,
            const FormulaT & _origin )  [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.43.4.10 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.

0.15.43.4.11 allModels()

`const std::vector<Model>& smtrat::Module::allModels ( ) const [inline], [inherited]`

Returns

All satisfying assignments, if existent.

0.15.43.4.12 anAnswerFound()

`bool smtrat::Module::anAnswerFound ( ) const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.43.4.13 answerFound()

`const smtrat::Conditionals& smtrat::Module::answerFound ( ) const [inline], [inherited]`

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.43.4.14 appliedPreprocessing()

`bool smtrat::PModule::appliedPreprocessing ( ) const [inline], [inherited]`

Returns

true, if the current received formula has been simplified and the result of this simplification is stored in the passed formula.

0.15.43.4.15 backendsModel()

`const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.43.4.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.43.4.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.43.4.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline],
[protected], [inherited]
```

0.15.43.4.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.43.4.20 check()

```
Answer smtrat::PModule::check (
    bool _final = false,
    bool _full = true,
    carl::Variable _objective = carl::Variable::NO_VARIABLE ) [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.43.4.21 checkCore()

`template<typename Settings >`
`Answer smtrat::BEModule< Settings >::checkCore ( ) [virtual]`

Checks the received formula for consistency.

Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.43.4.22 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.

0.15.43.4.23 checkModel()

`unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]`

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.43.4.24 cleanModel()

`void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.43.4.25 clearLemmas()

`void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

0.15.43.4.26 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

0.15.43.4.27 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

0.15.43.4.28 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.43.4.29 collectOrigins() [1/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

0.15.43.4.30 collectOrigins() [2/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inherited]`

Collects the formulas in the given formula, which are part of the received formula.
If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.43.4.31 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

0.15.43.4.32 constraintsToInform() `const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns

The constraints which the backends must be informed about.

0.15.43.4.33 currentlySatisfied() `virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.43.4.34 currentlySatisfiedByBackend()

```
unsigned smtrat::Module::currentlySatisfiedByBackend (
    const FormulaT & _formula ) const [inherited]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.43.4.35 deinform()

```
void smtrat::Module::deinform (
    const FormulaT & _constraint ) [inherited]
```

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.43.4.36 deinformCore()

```
virtual void smtrat::Module::deinformCore (
    [[maybe_unused] ] 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.

0.15.43.4.37 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

0.15.43.4.38 eraseSubformulaFromPassedFormula()

```
ModuleInput::iterator smtrat::Module::erase↩
SubformulaFromPassedFormula (
    ModuleInput::iterator _subformula,
    bool _ignoreOrigins = false ) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_subformula	The sub-formula to remove from the passed formula.

Parameters

_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 >.

0.15.43.4.39 excludeNotReceivedVariablesFromModel()

`void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

0.15.43.4.40 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

0.15.43.4.41 firstSubformulaToPass()

`ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.43.4.42 firstUncheckedReceivedSubformula()

`ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.43.4.43 freeSplittingVariable()

```
static void smtrat::Module::freeSplittingVariable (
    const FormulaT & _splittingVariable ) [inline], [static], [inherited]
```

0.15.43.4.44 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.43.4.45 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.43.4.46 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.43.4.47 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.43.4.48 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.

0.15.43.4.49 getInformationRelevantFormulas()

```
const std::set< FormulaT > & smtrat::Module↩
::getInformationRelevantFormulas ( ) [protected], [inherited]
```

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.43.4.50 getModelEqualities()

```
std::list< std::vector< carl::Variable > > smtrat::Module↩
::getModelEqualities ( ) const [virtual], [inherited]
```

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.43.4.51 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
            const FormulaT & _formula,
            FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.43.4.52 getOrigins() [2/3] void smtrat::Module::getOrigins (const FormulaT & _formula, FormulasT & _origins) const [inline], [protected], [inherited]

Parameters

_formula	
_origins	

0.15.43.4.53 getOrigins() [3/3] const FormulaT& smtrat::Module::getOrigins (ModuleInput::const_iterator _formula) const [inline], [protected], [inherited]

Gets the origins of the passed formula at the given position.

Parameters

_formula	The position of a formula in the passed formulas.

Returns

The origins of the passed formula at the given position.

0.15.43.4.54 getReceivedFormulaSimplified() std::pair< bool, FormulaT > smtrat::PModule::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 from smtrat::Module.

0.15.43.4.55 hasLemmas() bool smtrat::Module::hasLemmas ( ) [inline], [inherited]

Checks whether this module or any of its backends provides any lemmas.

0.15.43.4.56 hasValidInfeasibleSubset() bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.43.4.57 id() std::size_t smtrat::Module::id ( ) const [inline], [inherited]

Returns

A unique ID to identify this module instance.

0.15.43.4.58 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasible↩ Subsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.43.4.59 inform() `bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.
It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

Parameters

_constraint	The constraint to inform about.

Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

0.15.43.4.60 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.

0.15.43.4.61 informCore() `virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.43.4.62 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.

0.15.43.4.63 init() `void smtrat::Module::init ( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not be called twice and only before the first runBackends call.
Reimplemented in smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntBlastModule< Settings >, smtrat::FPPModule< Settings >, smtrat::CurryModule< Settings >, and smtrat::BVModule< Settings >.

0.15.43.4.64 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.43.4.65 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.

0.15.43.4.66 isPreprocessor() `bool smtrat::PModule::isPreprocessor ( ) const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.43.4.67 lemmas() `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.43.4.68 merge() `std::vector< FormulaT > smtrat::Module::merge ( const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.
({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

Parameters

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns

The vector being the two given vectors merged.

0.15.43.4.69 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.43.4.70 modelsDisjoint()

```
bool smtrat::Module::modelsDisjoint (
    const Model & _modelA,
    const Model & _modelB ) [static], [protected], [inherited]
```

Checks whether there is no variable assigned by both models.

Parameters

_modelA	The first model to check for.
_modelB	The second model to check for.

Returns

true, if there is no variable assigned by both models; false, otherwise.

0.15.43.4.71 moduleName()

```
template<typename Settings >
std::string smtrat::BEModule< Settings >::moduleName ( ) const [inline], [virtual]
```

Returns

The name of the given module type as name.

Reimplemented from smtrat::Module.

0.15.43.4.72 objective()

```
carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]
```

0.15.43.4.73 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
    const FormulaT & _origin ) const [protected], [inherited]
```

0.15.43.4.74 passedFormulaBegin()

```
ModuleInput::iterator smtrat::Module::passedFormulaBegin ( )
[inline], [protected], [inherited]
```

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.43.4.75 passedFormulaEnd()

```
ModuleInput::iterator smtrat::Module::passedFormulaEnd ( )
[inline], [protected], [inherited]
```

Returns

An iterator to the end of the passed formula.

0.15.43.4.76 pPassedFormula()

```
const ModuleInput* smtrat::Module::pPassedFormula ( ) const
[inline], [inherited]
```

Returns

A pointer to the formula passed to the backends of this module.

0.15.43.4.77 pReceivedFormula() `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.43.4.78 print() `void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.43.4.79 printAllModels() `void smtrat::Module::printAllModels ( std::ostream & _out = std::cout ) [inherited]`

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.43.4.80 printInfeasibleSubsets() `void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.43.4.81 printModel() `void smtrat::Module::printModel ( std::ostream & _out = std::cout ) const [inherited]`

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.43.4.82 printPassedFormula() `void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.43.4.83 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.43.4.84 probablyLooping()

`bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.43.4.85 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.43.4.86 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

0.15.43.4.87 receivedVariable()

`bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

0.15.43.4.88 remove()

`void smtrat::PModule::remove ( ModuleInput::const_iterator _subformula ) [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.
However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

Parameters

_subformula	The sub formula of the received formula to remove.

Reimplemented from smtrat::Module.

0.15.43.4.89 removeCore()

```
virtual void smtrat::Module::removeCore (
    [[maybe_unused] ] 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.

0.15.43.4.90 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin
(
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

0.15.43.4.91 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

0.15.43.4.92 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.43.4.93 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to this module.

0.15.43.4.94 runBackends() [1/2]

```
virtual Answer smtrat::PModule::runBackends ( ) [inline],
[virtual], [inherited]
```

Reimplemented from smtrat::Module.

0.15.43.4.95 runBackends() [2/2]

```
Answer smtrat::PModule::runBackends (
    bool _final,
    bool _full,
    carl::Variable _objective ) [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

Parameters

_final	true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT.
_full	false, if this module should avoid too expensive procedures and rather return unknown instead.
_objective	if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good.

Returns

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.43.4.96 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.

0.15.43.4.97 setThreadPriority()

`void smtrat::Module::setThreadPriority ( thread_priority _threadPriority ) [inline], [inherited]`

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.43.4.98 solverState()

`Answer smtrat::Module::solverState ( ) const [inline], [inherited]`

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.43.4.99 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 !=.

0.15.43.4.100 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.43.4.101 updateAllModels() `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.43.4.102 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.43.4.103 updateModel() `void smtrat::PModule::updateModel ( ) const [virtual], [inherited]`

Updates the current assignment into the model.
Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.
Reimplemented from smtrat::Module.

0.15.43.4.104 usedBackends() `const std::vector<Module*>& smtrat::Module::usedBackends ( ) const [inline], [inherited]`

Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.43.5 Field Documentation

0.15.43.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.43.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.43.5.3 mBackendsFoundAnswer `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.43.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.43.5.5 mFinalCheck `bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.43.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.43.5.7 mFirstSubformulaToPass

`ModuleInput::iterator smtrat::Module::mFirstSubformulaToPass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.43.5.8 mFirstUncheckedReceivedSubformula

`ModuleInput::const_iterator smtrat::Module::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.43.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.43.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.43.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.43.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.43.5.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.43.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.43.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.43.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.43.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.43.5.18 mObjectiveVariable `carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.43.5.19 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

0.15.43.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.43.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.43.5.22 mSolverState `std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.43.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

0.15.43.5.24 mUsedBackends `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.43.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.44 benchmax::BenchmarkResult Struct Reference

Results for a single benchmark run.

`#include <BenchmarkResult.h>`

Public Member Functions

- template<typename TimeLimit >
  void cleanup (const TimeLimit &limit)

  Properly detect timeouts.
- auto get_path () const
- void store (size_t id) const
- void store () const
- void restore () const
- void forget () const

Data Fields

- int exitCode

  Shell exit code.
- std::chrono::milliseconds time

  Runtime in milliseconds.
- std::string answer

  Answer string.
- std::size_t peak_memory_kbytes

  Peak memory usage.
- std::string stdout

  Standard output (mostly for parsing the answer and additional information).
- std::string stderr

  Error output (mostly for parsing the answer and additional information).
- std::map< std::string, std::string > additional

  Arbitrary additional information that can be provided by the tool class.
- size_t stored_id = 0

  Identifier for temporary file.

0.15.44.1 Detailed Description

Results for a single benchmark run.

0.15.44.2 Member Function Documentation

0.15.44.2.1 cleanup() `template<typename TimeLimit >`

```
void benchmax::BenchmarkResult::cleanup (
            const TimeLimit & limit )  [inline]
```

Properly detect timeouts.
Most backends give processes a bit more time to avoid having race-condition-like situations.

0.15.44.2.2 forget() `void benchmax::BenchmarkResult::forget ( ) const  [inline]`

0.15.44.2.3 get_path() `auto benchmax::BenchmarkResult::get_path ( ) const  [inline]`

0.15.44.2.4 restore() `void benchmax::BenchmarkResult::restore ( ) const  [inline]`

0.15.44.2.5 store() [1/2] `void benchmax::BenchmarkResult::store ( ) const  [inline]`

0.15.44.2.6 store() [2/2]

```
void benchmax::BenchmarkResult::store (
            size_t id ) const  [inline]
```

0.15.44.3 Field Documentation

0.15.44.3.1 additional `std::map<std::string, std::string> benchmax::BenchmarkResult::additional [mutable]`

Arbitrary additional information that can be provided by the tool class.

0.15.44.3.2 answer `std::string benchmax::BenchmarkResult::answer`

Answer string.

0.15.44.3.3 exitCode `int benchmax::BenchmarkResult::exitCode`

Shell exit code.

0.15.44.3.4 peak_memory_kbytes `std::size_t benchmax::BenchmarkResult::peak_memory_kbytes`

Peak memory usage.

0.15.44.3.5 stderr `std::string benchmax::BenchmarkResult::stderr`

Error output (mostly for parsing the answer and additional information).

0.15.44.3.6 stdout `std::string benchmax::BenchmarkResult::stdout`

Standard output (mostly for parsing the answer and additional information).

0.15.44.3.7 stored_id `size_t benchmax::BenchmarkResult::stored_id = 0 [mutable]`

Identifier for temporary file.

0.15.44.3.8 time `std::chrono::milliseconds benchmax::BenchmarkResult::time`

Runtime in milliseconds.

0.15.45 benchmax::BenchmarkSet Class Reference

A set of benchmarks from some common base directory.

`#include <BenchmarkSet.h>`

Public Member Functions

- void add_directory (const std::filesystem::path &dir)

  Recursively find all benchmarks from this directory.
- std::size_t size () const

  Number of files.
- auto begin () const

  Begin iterator.
- auto end () const

  End iterator.

0.15.45.1 Detailed Description

A set of benchmarks from some common base directory.

0.15.45.2 Member Function Documentation

0.15.45.2.1 add_directory() `void benchmax::BenchmarkSet::add_directory ( const std::filesystem::path & dir )`

Recursively find all benchmarks from this directory.

0.15.45.2.2 begin() `auto benchmax::BenchmarkSet::begin ( ) const [inline]`

Begin iterator.

0.15.45.2.3 end() `auto benchmax::BenchmarkSet::end ( ) const [inline]`

End iterator.

0.15.45.2.4 size() `std::size_t benchmax::BenchmarkSet::size ( ) const [inline]`

Number of files.

0.15.46 benchmax::settings::BenchmarkSettings Struct Reference

Settings for benchmarks.

`#include <benchmarks.h>`

Data Fields

- carl::settings::binary_quantity limit_memory

  Memory limit in megabytes.
- carl::settings::duration limit_time

  Time limit in seconds.
- carl::settings::duration grace_time

  Grace time in seconds.
- std::vector< std::filesystem::path > input_directories

  Lift of input directories.
- std::filesystem::path input_directories_common_prefix

  Common prefix of input directories (to shorten filenames in output).
- std::filesystem::path output_dir

  Output directory.
- std::filesystem::path output_file_xml

  Filename of xml file.

0.15.46.1 Detailed Description

Settings for benchmarks.

0.15.46.2 Field Documentation

0.15.46.2.1 grace_time `carl::settings::duration benchmax::settings::BenchmarkSettings::grace_time`

Grace time in seconds.

0.15.46.2.2 input_directories `std::vector<std::filesystem::path> benchmax::settings::BenchmarkSettings::input_directories`

Lift of input directories.

0.15.46.2.3 input_directories_common_prefix `std::filesystem::path benchmax::settings::BenchmarkSettings::input_directories_common_prefix`

Common prefix of input directories (to shorten filenames in output).

0.15.46.2.4 limit_memory `carl::settings::binary_quantity benchmax::settings::BenchmarkSettings::limit_memory`

Memory limit in megabytes.

0.15.46.2.5 limit_time `carl::settings::duration benchmax::settings::BenchmarkSettings::limit_time`

Time limit in seconds.

0.15.46.2.6 output_dir `std::filesystem::path benchmax::settings::BenchmarkSettings::output_dir`

Output directory.

0.15.46.2.7 output_file_xml `std::filesystem::path benchmax::settings::BenchmarkSettings::output_file_xml`

Filename of xml file.

0.15.47 smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName > Class Template Reference

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

`#include <Bimap.h>`

Public Types

- typedef std::forward_list< Class > Data
- typedef Data::iterator Iterator
- typedef Data::const_iterator ConstIterator

Public Member Functions

- Iterator begin () noexcept
- ConstIterator begin () const noexcept
- Iterator end () noexcept
- ConstIterator end () const noexcept
- Class & firstAt (const FirstKeyType &firstKey)
- Class & secondAt (const SecondKeyType &secondKey)
- Iterator firstFind (const FirstKeyType &firstKey)
- Iterator secondFind (const SecondKeyType &secondKey)
- template<typename... Args>
  Iterator emplace (Args &&... args)

0.15.47.1 Detailed Description

template<class Class, typename FirstKeyType, FirstKeyType Class::* FirstKeyName, typename SecondKeyType, SecondKeyType Class::* SecondKeyName>
class smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >

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

0.15.47.2 Member Typedef Documentation

0.15.47.2.1 ConstIterator

template<class Class , typename FirstKeyType , FirstKeyType Class::* FirstKeyName, typename SecondKeyType , SecondKeyType Class::* SecondKeyName>
typedef Data::const_iterator smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >::ConstIterator

0.15.47.2.2 Data

template<class Class , typename FirstKeyType , FirstKeyType Class::* First↩ KeyName, typename SecondKeyType , SecondKeyType Class::* SecondKeyName>
typedef std::forward_list<Class> smtrat::Bimap< Class, FirstKeyType, FirstKeyName, Second↩ KeyType, SecondKeyName >::Data

0.15.47.2.3 Iterator

template<class Class , typename FirstKeyType , FirstKeyType Class::* First↩ KeyName, typename SecondKeyType , SecondKeyType Class::* SecondKeyName>
typedef Data::iterator smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, Second↩ KeyName >::Iterator

0.15.47.3 Member Function Documentation

0.15.47.3.1 begin() [1/2]

template<class Class , typename FirstKeyType , FirstKeyType Class::* FirstKeyName, typename SecondKeyType , SecondKeyType Class::* SecondKeyName>
ConstIterator smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >::begin ( ) const [inline], [noexcept]

0.15.47.3.2 begin() [2/2]

template<class Class , typename FirstKeyType , FirstKeyType Class::* FirstKeyName, typename SecondKeyType , SecondKeyType Class::* SecondKeyName>
Iterator smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >↩ ::begin ( ) [inline], [noexcept]

0.15.47.3.3 emplace()

template<class Class , typename FirstKeyType , FirstKeyType Class::* FirstKeyName, typename SecondKeyType , SecondKeyType Class::* SecondKeyName>
template<typename... Args>
Iterator smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >↩ ::emplace (
Args &&... *args*) [inline]

0.15.47.3.4 end() [1/2]

template<class Class , typename FirstKeyType , FirstKeyType Class::* FirstKeyName, typename SecondKeyType , SecondKeyType Class::* SecondKeyName>
ConstIterator smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >::end ( ) const [inline], [noexcept]

0.15.47.3.5 end() [2/2]

template<class Class , typename FirstKeyType , FirstKeyType Class::* FirstKeyName, typename SecondKeyType , SecondKeyType Class::* SecondKeyName>
Iterator smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >::end ( ) [inline], [noexcept]

0.15.47.3.6 firstAt()

template<class Class , typename FirstKeyType , FirstKeyType Class::* First↩ KeyName, typename SecondKeyType , SecondKeyType Class::* SecondKeyName>

```
Class& smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >↩
::firstAt (
            const FirstKeyType & firstKey )  [inline]
```

0.15.47.3.7 firstFind()

```
template<class Class , typename FirstKeyType , FirstKeyType Class::*
FirstKeyName, typename SecondKeyType , SecondKeyType Class::* SecondKeyName>
Iterator smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >↩
::firstFind (
            const FirstKeyType & firstKey )  [inline]
```

0.15.47.3.8 secondAt()

```
template<class Class , typename FirstKeyType , FirstKeyType Class::*
FirstKeyName, typename SecondKeyType , SecondKeyType Class::* SecondKeyName>
Class& smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >↩
::secondAt (
            const SecondKeyType & secondKey )  [inline]
```

0.15.47.3.9 secondFind()

```
template<class Class , typename FirstKeyType , FirstKeyType Class::*
FirstKeyName, typename SecondKeyType , SecondKeyType Class::* SecondKeyName>
Iterator smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >↩
::secondFind (
            const SecondKeyType & secondKey )  [inline]
```

0.15.48 smtrat::expression::BinaryExpression Struct Reference

`#include <ExpressionContent.h>`

Public Member Functions

- BinaryExpression (BinaryType _type, Expression &&_lhs, Expression &&_rhs)

Data Fields

- BinaryType type
- Expression lhs
- Expression rhs

0.15.48.1 Constructor & Destructor Documentation

0.15.48.1.1 BinaryExpression()

```
smtrat::expression::BinaryExpression::BinaryExpression (
            BinaryType _type,
            Expression && _lhs,
            Expression && _rhs )  [inline]
```

0.15.48.2 Field Documentation

0.15.48.2.1 lhs

`Expression smtrat::expression::BinaryExpression::lhs`

0.15.48.2.2 rhs

`Expression smtrat::expression::BinaryExpression::rhs`

0.15.48.2.3 type `BinaryType smtrat::expression::BinaryExpression::type`

0.15.49 smtrat::parser::BinaryParser Struct Reference

Parses binaries: `#b[01]+`
`#include <Lexicon.h>`

Public Types

- typedef boost::iterator_range< Iterator > ITRange

Public Member Functions

- BinaryParser ()
- FixedWidthConstant< Integer > build (const ITRange &itr, const Integer &val)

Data Fields

- qi::uint_parser< Integer, 2, 1,-1 > number
- qi::rule< Iterator, FixedWidthConstant< Integer >), Skipper, qi::locals< Integer > > main
- qi::rule< Iterator, FixedWidthConstant< Integer >), Skipper > main2

0.15.49.1 Detailed Description

Parses binaries: `#b[01]+`

0.15.49.2 Member Typedef Documentation

0.15.49.2.1 ITRange `typedef boost::iterator_range<Iterator> smtrat::parser::BinaryParser::ITRange`

0.15.49.3 Constructor & Destructor Documentation

0.15.49.3.1 BinaryParser() `smtrat::parser::BinaryParser::BinaryParser ( ) [inline]`

0.15.49.4 Member Function Documentation

0.15.49.4.1 build()

```
FixedWidthConstant<Integer> smtrat::parser::BinaryParser::build (
    const ITRange & itr,
    const Integer & val ) [inline]
```

0.15.49.5 Field Documentation

0.15.49.5.1 main `qi::rule<Iterator, FixedWidthConstant<Integer>), Skipper, qi::locals<Integer> > smtrat::parser::BinaryParser::main`

0.15.49.5.2 main2 `qi::rule<Iterator, FixedWidthConstant<Integer>), Skipper> smtrat::parser↩ ::BinaryParser::main2`

0.15.49.5.3 number `qi::uint_parser<Integer,2,1,-1> smtrat::parser::BinaryParser::number`

0.15.50 smtrat::parser::BitvectorTheory Struct Reference

Implements the theory of bitvectors.
`#include <Bitvector.h>`

Public Types

- using OperatorType = carl::BVTermType

Public Member Functions

- BitvectorTheory (ParserState *state)
- bool declareVariable (const std::string &name, const carl::Sort &sort, types::VariableType &result, TheoryError &errors)

  Declare a new variable with the given name and the given sort.
- bool resolveSymbol (const Identifier &identifier, types::TermType &result, TheoryError &errors)

  Resolve a symbol that was not declared within the ParserState.
- bool handleITE (const FormulaT &ifterm, const types::TermType &thenterm, const types::TermType &elseterm, types::TermType &result, TheoryError &errors)

  Resolve an if-then-else operator.
- bool handleDistinct (const std::vector< types::TermType > &arguments, types::TermType &result, TheoryError &errors)

  Resolve a distinct operator.
- bool instantiate (const types::VariableType &var, const types::TermType &replacement, types::TermType &subject, TheoryError &errors)

  Instantiate a variable within a term.
- bool refreshVariable (const types::VariableType &var, types::VariableType &subject, TheoryError &errors)
- bool functionCall (const Identifier &identifier, const std::vector< types::TermType > &arguments, types::TermType &result, TheoryError &errors)

  Resolve another unknown function call.
- template<typename T , typename Builder >
  FormulaT expandDistinct (const std::vector< T > &values, const Builder &neqBuilder)

Static Public Member Functions

- static void addSimpleSorts (qi::symbols< char, carl::Sort > &sorts)
- static void addConstants (qi::symbols< char, types::ConstType > &)

  Initialize the global symbol table for constants.

Data Fields

- carl::Sort bvSort
- conversion::VectorVariantConverter< types::BVTerm > vectorConverter
- conversion::VariantConverter< types::BVTerm > termConverter
- ParserState * state

0.15.50.1 Detailed Description

Implements the theory of bitvectors.

0.15.50.2 Member Typedef Documentation

0.15.50.2.1 OperatorType

```
using smtrat::parser::BitvectorTheory::OperatorType = carl::BVTerm↩
Type
```

0.15.50.3 Constructor & Destructor Documentation

0.15.50.3.1 BitvectorTheory()

```
smtrat::parser::BitvectorTheory::BitvectorTheory (
    ParserState * state )
```

0.15.50.4 Member Function Documentation

0.15.50.4.1 addConstants()

```
static void smtrat::parser::AbstractTheory::addConstants (
    qi::symbols< char, types::ConstType > & ) [inline], [static], [inherited]
```

Initialize the global symbol table for constants.

0.15.50.4.2 addSimpleSorts()

```
void smtrat::parser::BitvectorTheory::addSimpleSorts (
    qi::symbols< char, carl::Sort > & sorts ) [static]
```

0.15.50.4.3 declareVariable()

```
bool smtrat::parser::BitvectorTheory::declareVariable (
    const std::string & ,
    const carl::Sort & ,
    types::VariableType & ,
    TheoryError & errors ) [virtual]
```

Declare a new variable with the given name and the given sort.
Reimplemented from smtrat::parser::AbstractTheory.

0.15.50.4.4 expandDistinct()

```
template<typename T , typename Builder >
FormulaT smtrat::parser::AbstractTheory::expandDistinct (
    const std::vector< T > & values,
    const Builder & neqBuilder ) [inline], [inherited]
```

0.15.50.4.5 functionCall()

```
bool smtrat::parser::BitvectorTheory::functionCall (
    const Identifier & ,
    const std::vector< types::TermType > & ,
    types::TermType & ,
    TheoryError & errors ) [virtual]
```

Resolve another unknown function call.
Reimplemented from smtrat::parser::AbstractTheory.

0.15.50.4.6 handleDistinct()

```
bool smtrat::parser::BitvectorTheory::handleDistinct (
    const std::vector< types::TermType > & ,
    types::TermType & ,
    TheoryError & errors ) [virtual]
```

Resolve a distinct operator.
Reimplemented from smtrat::parser::AbstractTheory.

0.15.50.4.7 handleITE()

```
bool smtrat::parser::BitvectorTheory::handleITE (
    const FormulaT & ,
    const types::TermType & ,
    const types::TermType & ,
    types::TermType & ,
    TheoryError & errors ) [virtual]
```

Resolve an if-then-else operator.
Reimplemented from smtrat::parser::AbstractTheory.

0.15.50.4.8 instantiate()

```
bool smtrat::parser::BitvectorTheory::instantiate (
    const types::VariableType & ,
    const types::TermType & ,
    types::TermType & ,
    TheoryError & errors ) [virtual]
```

Instantiate a variable within a term.
Reimplemented from smtrat::parser::AbstractTheory.

0.15.50.4.9 refreshVariable()

```
bool smtrat::parser::BitvectorTheory::refreshVariable (
    const types::VariableType & var,
    types::VariableType & subject,
    TheoryError & errors ) [virtual]
```

Reimplemented from smtrat::parser::AbstractTheory.

0.15.50.4.10 resolveSymbol()

```
bool smtrat::parser::BitvectorTheory::resolveSymbol (
    const Identifier & ,
    types::TermType & ,
    TheoryError & errors ) [virtual]
```

Resolve a symbol that was not declared within the ParserState.
This might be a symbol that actually uses indices, for example bitvector constants.
Reimplemented from smtrat::parser::AbstractTheory.

0.15.50.5 Field Documentation

0.15.50.5.1 bvSort

`carl::Sort smtrat::parser::BitvectorTheory::bvSort`

0.15.50.5.2 state

`ParserState* smtrat::parser::AbstractTheory::state [inherited]`

0.15.50.5.3 termConverter

`conversion::VariantConverter<types::BVTerm> smtrat::parser::BitvectorTheory::termConverter`

0.15.50.5.4 vectorConverter

`conversion::VectorVariantConverter<types::BVTerm> smtrat::parser::BitvectorTheory::vectorConverter`

0.15.51 smtrat::parser::types::BitvectorTheory Struct Reference

Types of the theory of bitvectors.

```
#include <TheoryTypes.h>
```

Public Types

- typedef mpl::vector< BVVariable, FixedWidthConstant< Integer >, BVTerm > ConstTypes
- typedef mpl::vector< BVVariable > VariableTypes
- typedef mpl::vector< BVVariable, FixedWidthConstant< Integer >, BVTerm, BVConstraint > ExpressionTypes
- typedef mpl::vector< BVVariable, FixedWidthConstant< Integer >, BVTerm, BVConstraint > TermTypes
- typedef carl::mpl_variant_of< TermTypes >::type TermType

0.15.51.1 Detailed Description

Types of the theory of bitvectors.

0.15.51.2 Member Typedef Documentation

0.15.51.2.1 ConstTypes `typedef mpl::vector<BVVariable, FixedWidthConstant<Integer>, BVTerm> smtrat::parser::types::BitvectorTheory::ConstTypes`

0.15.51.2.2 ExpressionTypes `typedef mpl::vector<BVVariable, FixedWidthConstant<Integer>, BVTerm, BVConstraint> smtrat::parser::types::BitvectorTheory::ExpressionTypes`

0.15.51.2.3 TermType `typedef carl::mpl_variant_of<TermTypes>::type smtrat::parser::types::BitvectorTheory::Te`

0.15.51.2.4 TermTypes `typedef mpl::vector<BVVariable, FixedWidthConstant<Integer>, BVTerm, BVConstraint> smtrat::parser::types::BitvectorTheory::TermTypes`

0.15.51.2.5 VariableTypes `typedef mpl::vector<BVVariable> smtrat::parser::types::BitvectorTheory::VariableType`

0.15.52 smtrat::BlastedConstr Class Reference

`#include <IntBlastModule.h>`

Public Member Functions

- BlastedConstr ()
- BlastedConstr (const FormulaT &_formula, const FormulasT &_constraints)
- BlastedConstr (const FormulaT &_formula)
- BlastedConstr (bool _satisfied)
- const FormulaT & formula () const
- const FormulasT & constraints () const

Friends

- std::ostream & operator<< (std::ostream &_out, const BlastedConstr &_constr)

0.15.52.1 Constructor & Destructor Documentation

0.15.52.1.1 BlastedConstr() [1/4] `smtrat::BlastedConstr::BlastedConstr ( ) [inline]`

0.15.52.1.2 BlastedConstr() [2/4]

```
smtrat::BlastedConstr::BlastedConstr (
    const FormulaT & _formula,
    const FormulasT & _constraints ) [inline]
```

0.15.52.1.3 BlastedConstr() [3/4]

```
smtrat::BlastedConstr::BlastedConstr (
    const FormulaT & _formula ) [inline]
```

0.15.52.1.4 BlastedConstr() [4/4]

```
smtrat::BlastedConstr::BlastedConstr (
    bool _satisfied ) [inline]
```

0.15.52.2 Member Function Documentation

0.15.52.2.1 constraints()

```
const FormulasT& smtrat::BlastedConstr::constraints ( ) const [inline]
```

0.15.52.2.2 formula()

```
const FormulaT& smtrat::BlastedConstr::formula ( ) const [inline]
```

0.15.52.3 Friends And Related Function Documentation

0.15.52.3.1 operator<<

```
std::ostream& operator<< (
    std::ostream & _out,
    const BlastedConstr & _constr ) [friend]
```

0.15.53 smtrat::BlastedPoly Class Reference

```
#include <IntBlastModule.h>
```

Public Member Functions

- BlastedPoly ()
- BlastedPoly (Integer _constant)
- BlastedPoly (Integer _constant, FormulasT _constraints)
- BlastedPoly (AnnotatedBVTerm _term)
- BlastedPoly (AnnotatedBVTerm _term, FormulasT _constraints)
- bool is_constant () const
- const Integer & constant () const
- const AnnotatedBVTerm & term () const
- const FormulasT & constraints () const
- const Integer & lower_bound () const
- const Integer & upper_bound () const

Friends

- std::ostream & operator<< (std::ostream &_out, const BlastedPoly &_poly)

0.15.53.1 Constructor & Destructor Documentation

0.15.53.1.1 BlastedPoly() [1/5]

```
smtrat::BlastedPoly::BlastedPoly ( ) [inline]
```

0.15.53.1.2 BlastedPoly() [2/5]

```
smtrat::BlastedPoly::BlastedPoly (
    Integer _constant ) [inline]
```

0.15.53.1.3 BlastedPoly() [3/5]

```
smtrat::BlastedPoly::BlastedPoly (
    Integer _constant,
    FormulasT _constraints ) [inline]
```

0.15.53.1.4 BlastedPoly() [4/5]

```
smtrat::BlastedPoly::BlastedPoly (
    AnnotatedBVTerm _term ) [inline]
```

0.15.53.1.5 BlastedPoly() [5/5]

```
smtrat::BlastedPoly::BlastedPoly (
    AnnotatedBVTerm _term,
    FormulasT _constraints ) [inline]
```

0.15.53.2 Member Function Documentation

0.15.53.2.1 constant()

```
const Integer& smtrat::BlastedPoly::constant ( ) const [inline]
```

0.15.53.2.2 constraints()

```
const FormulasT& smtrat::BlastedPoly::constraints ( ) const [inline]
```

0.15.53.2.3 is_constant()

```
bool smtrat::BlastedPoly::is_constant ( ) const [inline]
```

0.15.53.2.4 lower_bound()

```
const Integer& smtrat::BlastedPoly::lower_bound ( ) const [inline]
```

0.15.53.2.5 term()

```
const AnnotatedBVTerm& smtrat::BlastedPoly::term ( ) const [inline]
```

0.15.53.2.6 upper_bound()

```
const Integer& smtrat::BlastedPoly::upper_bound ( ) const [inline]
```

0.15.53.3 Friends And Related Function Documentation

0.15.53.3.1 operator<<

```
std::ostream& operator<< (
    std::ostream & _out,
    const BlastedPoly & _poly ) [friend]
```

0.15.54 smtrat::mcsat::Bookkeeping Class Reference

Represent the trail, i.e.

```
#include <Bookkeeping.h>
```

Public Member Functions

- const auto & model () const
- const auto & assignedVariables () const
- const auto & assignments () const
- const auto & constraints () const
- const auto & mvBounds () const
- const auto & variables () const
- void updateVariables (const carl::Variables &variables)
- void pushConstraint (const FormulaT &f)

  Assert a constraint/literal.
- void popConstraint (const FormulaT &f)
- void pushAssignment (carl::Variable v, const ModelValue &mv, const FormulaT &f)
- void popAssignment (carl::Variable v)

0.15.54.1 Detailed Description

Represent the trail, i.e.
the assignment/model state, of a MCSAT run in different representations (kept in sync) for a fast access. Most notably, we store literals, i.e. a polynomial (in)equality-atom or its negation, which we assert to be true. Since the negation can be represented by an atom by flipping the (in)equality, it suffices to store only atoms. Here we call atomic formulas over the theory of real arithmetic "constraints".

0.15.54.2 Member Function Documentation

0.15.54.2.1 assignedVariables() `const auto& smtrat::mcsat::Bookkeeping::assignedVariables ( ) const [inline]`

0.15.54.2.2 assignments() `const auto& smtrat::mcsat::Bookkeeping::assignments ( ) const [inline]`

0.15.54.2.3 constraints() `const auto& smtrat::mcsat::Bookkeeping::constraints ( ) const [inline]`

0.15.54.2.4 model() `const auto& smtrat::mcsat::Bookkeeping::model ( ) const [inline]`

0.15.54.2.5 mvBounds() `const auto& smtrat::mcsat::Bookkeeping::mvBounds ( ) const [inline]`

0.15.54.2.6 popAssignment()

```
void smtrat::mcsat::Bookkeeping::popAssignment (
    carl::Variable v ) [inline]
```

0.15.54.2.7 popConstraint()

```
void smtrat::mcsat::Bookkeeping::popConstraint (
    const FormulaT & f ) [inline]
```

0.15.54.2.8 pushAssignment()

```
void smtrat::mcsat::Bookkeeping::pushAssignment (
    carl::Variable v,
    const ModelValue & mv,
    const FormulaT & f ) [inline]
```

0.15.54.2.9 pushConstraint() `void smtrat::mcsat::Bookkeeping::pushConstraint ( const FormulaT & f ) [inline]`

Assert a constraint/literal.

0.15.54.2.10 updateVariables() `void smtrat::mcsat::Bookkeeping::updateVariables ( const carl::Variables & variables ) [inline]`

0.15.54.2.11 variables() `const auto& smtrat::mcsat::Bookkeeping::variables ( ) const [inline]`

0.15.55 smtrat::covering_ng::formula::node_ds::BOOL Struct Reference

`#include <FormulaEvaluation.h>`

Data Fields

- carl::Variable variable

0.15.55.1 Field Documentation

0.15.55.1.1 variable `carl::Variable smtrat::covering_ng::formula::node_ds::BOOL::variable`

0.15.56 smtrat::parser::BooleanEncodingTheory Struct Reference

Implements the theory of bitvectors.

`#include <BooleanEncoding.h>`

Public Member Functions

- BooleanEncodingTheory (ParserState *state)
- virtual bool declareVariable (const std::string &, const carl::Sort &, types::VariableType &, TheoryError &errors)

  Declare a new variable with the given name and the given sort.
- virtual bool resolveSymbol (const Identifier &, types::TermType &, TheoryError &errors)

  Resolve a symbol that was not declared within the ParserState.
- virtual bool handleITE (const FormulaT &, const types::TermType &, const types::TermType &, types::TermType &, TheoryError &errors)

  Resolve an if-then-else operator.
- virtual bool handleDistinct (const std::vector< types::TermType > &, types::TermType &, TheoryError &errors)

  Resolve a distinct operator.
- template<typename T , typename Builder >
  FormulaT expandDistinct (const std::vector< T > &values, const Builder &neqBuilder)
- virtual bool instantiate (const types::VariableType &, const types::TermType &, types::TermType &, TheoryError &errors)

  Instantiate a variable within a term.
- virtual bool refreshVariable (const types::VariableType &, types::VariableType &, TheoryError &errors)
- virtual bool functionCall (const Identifier &, const std::vector< types::TermType > &, types::TermType &, TheoryError &errors)

  Resolve another unknown function call.

Static Public Member Functions

- static void addSimpleSorts (qi::symbols< char, carl::Sort > &)

  Initialize the global symbol table for simple sorts.
- static void addConstants (qi::symbols< char, types::ConstType > &)

  Initialize the global symbol table for constants.

Data Fields

- ParserState * state

0.15.56.1 Detailed Description

Implements the theory of bitvectors.

0.15.56.2 Constructor & Destructor Documentation

0.15.56.2.1 BooleanEncodingTheory()

```
smtrat::parser::BooleanEncodingTheory::BooleanEncoding↩
Theory (
            ParserState * state )
```

0.15.56.3 Member Function Documentation

0.15.56.3.1 addConstants()

```
static void smtrat::parser::AbstractTheory::addConstants (
            qi::symbols< char, types::ConstType > & ) [inline], [static], [inherited]
```

Initialize the global symbol table for constants.

0.15.56.3.2 addSimpleSorts()

```
static void smtrat::parser::AbstractTheory::addSimpleSorts (
            qi::symbols< char, carl::Sort > & ) [inline], [static], [inherited]
```

Initialize the global symbol table for simple sorts.

0.15.56.3.3 declareVariable()

```
virtual bool smtrat::parser::AbstractTheory::declareVariable (
            const std::string & ,
            const carl::Sort & ,
            types::VariableType & ,
            TheoryError & errors ) [inline], [virtual], [inherited]
```

Declare a new variable with the given name and the given sort.
Reimplemented in smtrat::parser::UninterpretedTheory, smtrat::parser::CoreTheory, smtrat::parser::BitvectorTheory, and smtrat::parser::ArithmeticTheory.

0.15.56.3.4 expandDistinct()

```
template<typename T , typename Builder >
FormulaT smtrat::parser::AbstractTheory::expandDistinct (
            const std::vector< T > & values,
            const Builder & neqBuilder ) [inline], [inherited]
```

0.15.56.3.5 functionCall()

```
virtual bool smtrat::parser::AbstractTheory::functionCall (
    const Identifier & ,
    const std::vector< types::TermType > & ,
    types::TermType & ,
    TheoryError & errors )  [inline], [virtual], [inherited]
```

Resolve another unknown function call.
Reimplemented in smtrat::parser::UninterpretedTheory, smtrat::parser::CoreTheory, smtrat::parser::BitvectorTheory, and smtrat::parser::ArithmeticTheory.

0.15.56.3.6 handleDistinct()

```
virtual bool smtrat::parser::AbstractTheory::handleDistinct (
    const std::vector< types::TermType > & ,
    types::TermType & ,
    TheoryError & errors )  [inline], [virtual], [inherited]
```

Resolve a distinct operator.
Reimplemented in smtrat::parser::CoreTheory, smtrat::parser::BitvectorTheory, smtrat::parser::ArithmeticTheory, and smtrat::parser::UninterpretedTheory.

0.15.56.3.7 handleITE()

```
virtual bool smtrat::parser::AbstractTheory::handleITE (
    const FormulaT & ,
    const types::TermType & ,
    const types::TermType & ,
    types::TermType & ,
    TheoryError & errors )  [inline], [virtual], [inherited]
```

Resolve an if-then-else operator.
Reimplemented in smtrat::parser::UninterpretedTheory, smtrat::parser::CoreTheory, smtrat::parser::BitvectorTheory, and smtrat::parser::ArithmeticTheory.

0.15.56.3.8 instantiate()

```
virtual bool smtrat::parser::AbstractTheory::instantiate (
    const types::VariableType & ,
    const types::TermType & ,
    types::TermType & ,
    TheoryError & errors )  [inline], [virtual], [inherited]
```

Instantiate a variable within a term.
Reimplemented in smtrat::parser::BitvectorTheory, and smtrat::parser::ArithmeticTheory.

0.15.56.3.9 refreshVariable()

```
virtual bool smtrat::parser::AbstractTheory::refreshVariable (
    const types::VariableType & ,
    types::VariableType & ,
    TheoryError & errors )  [inline], [virtual], [inherited]
```

Reimplemented in smtrat::parser::BitvectorTheory.

0.15.56.3.10 resolveSymbol()

```
virtual bool smtrat::parser::AbstractTheory::resolveSymbol (
    const Identifier & ,
    types::TermType & ,
    TheoryError & errors )  [inline], [virtual], [inherited]
```

Resolve a symbol that was not declared within the ParserState.
This might be a symbol that actually uses indices, for example bitvector constants.
Reimplemented in smtrat::parser::BitvectorTheory.

0.15.56.4 Field Documentation

0.15.56.4.1 state `ParserState* smtrat::parser::AbstractTheory::state [inherited]`

0.15.57 Minisat::BoolOption Class Reference

`#include <Options.h>`

Public Member Functions

- BoolOption (const char *c, const char *n, const char *d, bool v)
- operator bool (void) const
- BoolOption & operator= (bool b)
- virtual void help (bool verbose=false)

Static Protected Member Functions

- static vec< Option * > & getOptionList ()
- static const char *& getUsageString ()
- static const char *& getHelpPrefixString ()

Protected Attributes

- const char * name
- const char * description
- const char * category
- const char * type_name

0.15.57.1 Constructor & Destructor Documentation

0.15.57.1.1 BoolOption()

```
Minisat::BoolOption::BoolOption (
    const char * c,
    const char * n,
    const char * d,
    bool v ) [inline]
```

0.15.57.2 Member Function Documentation

0.15.57.2.1 getHelpPrefixString()

`static const char*& Minisat::Option::getHelpPrefixString ( ) [inline], [static], [protected], [inherited]`

0.15.57.2.2 getOptionList()

`static vec<Option*>& Minisat::Option::getOptionList ( ) [inline], [static], [protected], [inherited]`

0.15.57.2.3 getUsageString()

`static const char*& Minisat::Option::getUsageString ( ) [inline], [static], [protected], [inherited]`

0.15.57.2.4 help()

```
virtual void Minisat::BoolOption::help (
    bool verbose = false ) [inline], [virtual]
```

Implements Minisat::Option.

0.15.57.2.5 operator bool() `Minisat::BoolOption::operator bool ( void ) const [inline]`

0.15.57.2.6 operator=() `BoolOption& Minisat::BoolOption::operator= ( bool b ) [inline]`

0.15.57.3 Field Documentation

0.15.57.3.1 category `const char* Minisat::Option::category [protected], [inherited]`

0.15.57.3.2 description `const char* Minisat::Option::description [protected], [inherited]`

0.15.57.3.3 name `const char* Minisat::Option::name [protected], [inherited]`

0.15.57.3.4 type_name `const char* Minisat::Option::type_name [protected], [inherited]`

0.15.58 smtrat::cadcells::datastructures::Bound Class Reference

Bound type of a SymbolicInterval.

`#include <roots.h>`

Public Member Functions

- bool is_infty () const
- bool is_strict () const
- bool is_weak () const
- const RootFunction & value () const
- void set_weak ()
- bool operator== (const Bound &other) const
- bool operator!= (const Bound &other) const

Static Public Member Functions

- static Bound infty ()
- static Bound strict (RootFunction value)
- static Bound weak (RootFunction value)

0.15.58.1 Detailed Description

Bound type of a SymbolicInterval.

0.15.58.2 Member Function Documentation

0.15.58.2.1 infty() `static Bound smtrat::cadcells::datastructures::Bound::infty ( ) [inline], [static]`

0.15.58.2.2 is_infty() `bool smtrat::cadcells::datastructures::Bound::is_infty ( ) const [inline]`

0.15.58.2.3 is_strict() `bool smtrat::cadcells::datastructures::Bound::is_strict ( ) const [inline]`

0.15.58.2.4 is_weak() `bool smtrat::cadcells::datastructures::Bound::is_weak ( ) const [inline]`

0.15.58.2.5 operator"!=() `bool smtrat::cadcells::datastructures::Bound::operator!= ( const Bound & other ) const [inline]`

0.15.58.2.6 operator==() `bool smtrat::cadcells::datastructures::Bound::operator== ( const Bound & other ) const [inline]`

0.15.58.2.7 set_weak() `void smtrat::cadcells::datastructures::Bound::set_weak ( ) [inline]`

0.15.58.2.8 strict() `static Bound smtrat::cadcells::datastructures::Bound::strict ( RootFunction value ) [inline], [static]`

0.15.58.2.9 value() `const RootFunction& smtrat::cadcells::datastructures::Bound::value ( ) const [inline]`

0.15.58.2.10 weak() `static Bound smtrat::cadcells::datastructures::Bound::weak ( RootFunction value ) [inline], [static]`

0.15.59 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)

0.15.59.1 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".

0.15.59.2 Member Typedef Documentation

0.15.59.2.1 BoundSet template<typename T1 , typename T2 >

```
typedef carl::PointerSet<Bound<T1, T2> > smtrat::lra::Bound< T1, T2 >::BoundSet
```

A set of pointer to bounds.

0.15.59.3 Member Enumeration Documentation

0.15.59.3.1 Type template<typename T1 , typename T2 >

```
enum smtrat::lra::Bound::Type
```

The type of the bound: LOWER = ">=", UPPER = "<=", EQUAL "==".

Enumerator

LOWER	
UPPER	
EQUAL	

0.15.59.4 Constructor & Destructor Documentation

0.15.59.4.1 Bound() [1/2] template<typename T1 , typename T2 >

```
smtrat::lra::Bound< T1, T2 >::Bound ( ) [delete]
```

0.15.59.4.2 Bound() [2/2] template<typename T1 , typename T2 >

```
smtrat::lra::Bound< T1, T2 >::Bound (
            const Value< T1 > *const _limit,
            Variable< T1, T2 > *const _var,
            Type _type,
            const FormulaT & _constraint,
            Bound< T1, T2 >::Info * _boundInfo = NULL,
            bool _deduced = false )
```

Parameters

_limit	
_var	
_type	
_constraint	
_boundInfo	
_deduced	

0.15.59.4.3 ~Bound() template<typename T1 , typename T2 >

```
smtrat::lra::Bound< T1, T2 >::~Bound ( )
```

0.15.59.5 Member Function Documentation

0.15.59.5.1 asConstraint() template<typename T1 , typename T2 >

```
const FormulaT& smtrat::lra::Bound< T1, T2 >::asConstraint ( ) const [inline]
```

Returns

0.15.59.5.2 boundExists()

`template<typename T1 , typename T2 >`
`void smtrat::lra::Bound< T1, T2 >::boundExists ( ) const [inline]`

0.15.59.5.3 deduced()

`template<typename T1 , typename T2 >`
`bool smtrat::lra::Bound< T1, T2 >::deduced ( ) const [inline]`

Returns

0.15.59.5.4 exists()

`template<typename T1 , typename T2 >`
`bool smtrat::lra::Bound< T1, T2 >::exists ( ) const [inline]`

0.15.59.5.5 hasNeqRepresentation()

`template<typename T1 , typename T2 >`
`bool smtrat::lra::Bound< T1, T2 >::hasNeqRepresentation ( ) const [inline]`

Returns

0.15.59.5.6 isActive()

`template<typename T1 , typename T2 >`
`bool smtrat::lra::Bound< T1, T2 >::isActive ( ) const [inline]`

Returns

0.15.59.5.7 isComplementActive()

`template<typename T1 , typename T2 >`
`bool smtrat::lra::Bound< T1, T2 >::isComplementActive ( ) const [inline]`

Returns

0.15.59.5.8 isInfinite()

`template<typename T1 , typename T2 >`
`bool smtrat::lra::Bound< T1, T2 >::isInfinite ( ) const [inline]`

Returns

0.15.59.5.9 isLowerBound()

`template<typename T1 , typename T2 >`
`bool smtrat::lra::Bound< T1, T2 >::isLowerBound ( ) const [inline]`

Returns

0.15.59.5.10 isSatisfied()

```
template<typename T1 , typename T2 >
bool smtrat::lra::Bound< T1, T2 >::isSatisfied (
            const T1 & _delta ) const  [inline]
```

0.15.59.5.11 isUnassigned()

```
template<typename T1 , typename T2 >
bool smtrat::lra::Bound< T1, T2 >::isUnassigned ( ) const  [inline]
```

0.15.59.5.12 isUpperBound()

```
template<typename T1 , typename T2 >
bool smtrat::lra::Bound< T1, T2 >::isUpperBound ( ) const  [inline]
```

Returns

0.15.59.5.13 isWeak()

```
template<typename T1 , typename T2 >
bool smtrat::lra::Bound< T1, T2 >::isWeak ( ) const  [inline]
```

Returns

0.15.59.5.14 limit()

```
template<typename T1 , typename T2 >
const Value<T1>& smtrat::lra::Bound< T1, T2 >::limit ( ) const  [inline]
```

Returns

0.15.59.5.15 markAsDeleted()

```
template<typename T1 , typename T2 >
void smtrat::lra::Bound< T1, T2 >::markAsDeleted ( ) const  [inline]
```

Returns

0.15.59.5.16 markedAsDeleted()

```
template<typename T1 , typename T2 >
bool smtrat::lra::Bound< T1, T2 >::markedAsDeleted ( ) const  [inline]
```

Returns

0.15.59.5.17 neqRepresentation()

```
template<typename T1 , typename T2 >
const FormulaT& smtrat::lra::Bound< T1, T2 >::neqRepresentation ( ) const  [inline]
```

Returns

0.15.59.5.18 operator<() [1/3]

```
template<typename T1 , typename T2 >
bool smtrat::lra::Bound< T1, T2 >::operator< (
            const Bound< T1, T2 > & _bound ) const
```

Parameters

_bound	

Returns

0.15.59.5.19 operator<() [2/3]

```
template<typename T1 , typename T2 >
bool smtrat::lra::Bound< T1, T2 >::operator< (
            const T1 & _a ) const
```

Parameters

↩ _↩ *a*	

Returns

0.15.59.5.20 operator<() [3/3]

```
template<typename T1 , typename T2 >
bool smtrat::lra::Bound< T1, T2 >::operator< (
            const Value< T1 > & _value ) const
```

Parameters

_value	

Returns

0.15.59.5.21 operator<=() [1/2]

```
template<typename T1 , typename T2 >
bool smtrat::lra::Bound< T1, T2 >::operator<= (
            const T1 & _a ) const
```

Parameters

↩ _↩ *a*	

Returns

0.15.59.5.22 operator<=() [2/2]

```
template<typename T1 , typename T2 >
bool smtrat::lra::Bound< T1, T2 >::operator<= (
            const Value< T1 > & v ) const  [inline]
```

Returns

0.15.59.5.23 operator==() [1/2]

```
template<typename T1 , typename T2 >
bool smtrat::lra::Bound< T1, T2 >::operator== (
            const T1 & _a ) const
```

Parameters

↩ _↩ *a*	

Returns

0.15.59.5.24 operator==() [2/2]

```
template<typename T1 , typename T2 >
bool smtrat::lra::Bound< T1, T2 >::operator== (
            const Value< T1 > & _value ) const
```

Parameters

_value	

Returns

0.15.59.5.25 operator>() [1/3]

```
template<typename T1 , typename T2 >
bool smtrat::lra::Bound< T1, T2 >::operator> (
            const Bound< T1, T2 > & _bound ) const
```

Parameters

_bound	

Returns

0.15.59.5.26 operator>() [2/3]

```
template<typename T1 , typename T2 >
bool smtrat::lra::Bound< T1, T2 >::operator> (
            const T1 & _a ) const
```

Parameters

↩ _↩ *a*	

Returns

0.15.59.5.27 operator>() [3/3]

```
template<typename T1 , typename T2 >
bool smtrat::lra::Bound< T1, T2 >::operator> (
            const Value< T1 > & _value ) const
```

Parameters

_value	

Returns

0.15.59.5.28 operator>=() [1/2]

```
template<typename T1 , typename T2 >
bool smtrat::lra::Bound< T1, T2 >::operator>= (
            const T1 & _a ) const
```

Parameters

↩ _↩ *a*	

Returns

0.15.59.5.29 operator>=() [2/2]

```
template<typename T1 , typename T2 >
bool smtrat::lra::Bound< T1, T2 >::operator>= (
            const Value< T1 > & v ) const  [inline]
```

Returns

0.15.59.5.30 origins()

```
template<typename T1 , typename T2 >
const std::vector<FormulaT>& smtrat::lra::Bound< T1, T2 >::origins ( ) const  [inline]
```

Returns

0.15.59.5.31 pInfo()

```
template<typename T1 , typename T2 >
Info* smtrat::lra::Bound< T1, T2 >::pInfo ( ) const  [inline]
```

Returns

0.15.59.5.32 pLimit()

```
template<typename T1 , typename T2 >
const Value<T1>* smtrat::lra::Bound< T1, T2 >::pLimit ( ) const  [inline]
```

Returns

0.15.59.5.33 pOrigins()

```
template<typename T1 , typename T2 >
const std::shared_ptr<std::vector<FormulaT> >& smtrat::lra::Bound< T1, T2 >::pOrigins ( )
const  [inline]
```

Returns

0.15.59.5.34 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	

0.15.59.5.35 pVariable()

```
template<typename T1 , typename T2 >
Variable<T1, T2>* smtrat::lra::Bound< T1, T2 >::pVariable ( ) const  [inline]
```

Returns

0.15.59.5.36 resetNeqRepresentation()

```
template<typename T1 , typename T2 >
void smtrat::lra::Bound< T1, T2 >::resetNeqRepresentation ( ) const  [inline]
```

0.15.59.5.37 setComplement()

```
template<typename T1 , typename T2 >
void smtrat::lra::Bound< T1, T2 >::setComplement (
            const Bound< T1, T2 > * _complement ) const  [inline]
```

0.15.59.5.38 setNeqRepresentation()

```
template<typename T1 , typename T2 >
void smtrat::lra::Bound< T1, T2 >::setNeqRepresentation (
            const FormulaT & _constraint ) const  [inline]
```

Parameters

_constraint	

0.15.59.5.39 toString()

```
template<typename T1 , typename T2 >
const std::string smtrat::lra::Bound< T1, T2 >::toString ( ) const
```

Returns

0.15.59.5.40 type()

```
template<typename T1 , typename T2 >
Type smtrat::lra::Bound< T1, T2 >::type ( ) const  [inline]
```

Returns

0.15.59.5.41 unmarkAsDeleted()

```
template<typename T1 , typename T2 >
void smtrat::lra::Bound< T1, T2 >::unmarkAsDeleted ( ) const  [inline]
```

Returns

0.15.59.5.42 variable()

```
template<typename T1 , typename T2 >
const Variable<T1, T2>& smtrat::lra::Bound< T1, T2 >::variable ( ) const  [inline]
```

Returns

0.15.59.6 Friends And Related Function Documentation

0.15.59.6.1 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	

0.15.60 smtrat::mcsat::fm::Bound Struct Reference

```
#include <ConflictGenerator.h>
```

Public Member Functions

- Bound (ConstraintT constr, Poly p, Poly q, Rational r, bool neg)

Data Fields

- ConstraintT constr
- Poly p
- Poly q
- Rational r
- bool neg

Friends

- std::ostream & operator<< (std::ostream &os, const Bound &dt)

0.15.60.1 Constructor & Destructor Documentation

0.15.60.1.1 Bound()

```
smtrat::mcsat::fm::Bound::Bound (
    ConstraintT constr,
    Poly p,
    Poly q,
    Rational r,
    bool neg ) [inline]
```

0.15.60.2 Friends And Related Function Documentation

0.15.60.2.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const Bound & dt ) [friend]
```

0.15.60.3 Field Documentation

0.15.60.3.1 constr

```
ConstraintT smtrat::mcsat::fm::Bound::constr
```

0.15.60.3.2 neg

```
bool smtrat::mcsat::fm::Bound::neg
```

0.15.60.3.3 p

```
Poly smtrat::mcsat::fm::Bound::p
```

0.15.60.3.4 q

```
Poly smtrat::mcsat::fm::Bound::q
```

0.15.60.3.5 r

```
Rational smtrat::mcsat::fm::Bound::r
```

0.15.61 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.

0.15.61.1 Detailed Description

template<typename T>
class smtrat::vb::Bound< T >

Class for the bound of a variable.

0.15.61.2 Member Enumeration Documentation

Enumerator

0.15.61.2.1 Type `template<typename T >`

`enum smtrat::vb::Bound::Type`

The type of a bounds.

Enumerator

STRICT_LOWER_BOUND	
WEAK_LOWER_BOUND	
EQUAL_BOUND	
WEAK_UPPER_BOUND	
STRICT_UPPER_BOUND	

0.15.61.3 Constructor & Destructor Documentation

0.15.61.3.1 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)

0.15.61.3.2 ~Bound() `template<typename T >`

`smtrat::vb::Bound< T >::~Bound ( )`

Destructs this bound.

0.15.61.4 Member Function Documentation

0.15.61.4.1 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.

0.15.61.4.2 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.

0.15.61.4.3 isActive() template<typename T >

```
bool smtrat::vb::Bound< T >::isActive ( ) const [inline]
```

Returns

true, if this bound is active, which means that there is at least one origin for it left. Note, that origins can be removed belatedly. false, otherwise.

0.15.61.4.4 isInfinite() template<typename T >

```
bool smtrat::vb::Bound< T >::isInfinite ( ) const [inline]
```

Returns

true, if the bound infinite; false, otherwise.

0.15.61.4.5 isLowerBound() template<typename T >

```
bool smtrat::vb::Bound< T >::isLowerBound ( ) const [inline]
```

Returns

true, if the bound is a lower bound; false, otherwise.

0.15.61.4.6 isUpperBound() template<typename T >

```
bool smtrat::vb::Bound< T >::isUpperBound ( ) const [inline]
```

Returns

true, if the bound is an upper bound; false, otherwise.

0.15.61.4.7 limit() template<typename T >

```
const Rational& smtrat::vb::Bound< T >::limit ( ) const [inline]
```

Returns

A constant reference to the value of the limit. Note, that it must be ensured that the bound is finite before calling this method.

0.15.61.4.8 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.

0.15.61.4.9 origins() `template<typename T >`

`const std::set<T,carl::less<T,false> >& smtrat::vb::Bound< T >::origins ( ) const [inline]`

Returns

A constant reference to the set of origins of this bound.

0.15.61.4.10 pLimit() `template<typename T >`

`const Rational* smtrat::vb::Bound< T >::pLimit ( ) const [inline]`

Returns

A pointer to the limit of this bound.

0.15.61.4.11 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.

0.15.61.4.12 pVariable() `template<typename T >`

`Variable<T>* smtrat::vb::Bound< T >::pVariable ( ) const [inline]`

Returns

A pointer to the variable wrapper considered by this bound.

0.15.61.4.13 type() `template<typename T >`

`Type smtrat::vb::Bound< T >::type ( ) const [inline]`

Returns

The type of this bound.

0.15.61.4.14 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.

0.15.61.5 Friends And Related Function Documentation

0.15.61.5.1 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.

0.15.62 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).

0.15.62.1 Detailed Description

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

0.15.62.2 Constructor & Destructor Documentation

0.15.62.2.1 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.

0.15.62.3 Field Documentation

0.15.62.3.1 mIncreasing

```
int smtrat::Branching::mIncreasing
```

Stores whether this the branching of the polynomial has been on an increasing sequence of values (>0), or on a decreasing sequence of values (<0).
It is initially zero.

0.15.62.3.2 mPolynomial

```
Poly::PolyType smtrat::Branching::mPolynomial
```

The polynomial to branch at.

0.15.62.3.3 mRepetitions

```
size_t smtrat::Branching::mRepetitions
```

The number of repetitions of the branching.

0.15.62.3.4 mValue

```
Rational smtrat::Branching::mValue
```

The value to branch the polynomial at.

0.15.63 smtrat::BVAnnotation Class Reference

```
#include <IntBlastModule.h>
```

Public Member Functions

- BVAnnotation ()
- BVAnnotation (std::size_t _width, bool _signed, Integer _offset=0)
- std::size_t width () const
- bool isSigned () const
- bool is_constant () const
- bool hasOffset () const
- BVAnnotation withOffset (Integer _newOffset) const
- BVAnnotation withWidth (std::size_t _width) const
- const Integer & offset () const
- const carl::Interval< Integer > & bounds () const
- const Integer & lower_bound () const
- const Integer & upper_bound () const

Static Public Member Functions

- static BVAnnotation forSum (BVAnnotation _summand1, BVAnnotation _summand2)
- static BVAnnotation forProduct (BVAnnotation _factor1, BVAnnotation _factor2)

Friends

- std::ostream & operator<< (std::ostream &_out, const BVAnnotation &_type)

0.15.63.1 Constructor & Destructor Documentation

0.15.63.1.1 BVAnnotation() [1/2]

```
smtrat::BVAnnotation::BVAnnotation ( ) [inline]
```

0.15.63.1.2 BVAnnotation() [2/2]

```
smtrat::BVAnnotation::BVAnnotation (
    std::size_t _width,
    bool _signed,
    Integer _offset = 0 ) [inline]
```

0.15.63.2 Member Function Documentation

0.15.63.2.1 bounds()

```
const carl::Interval<Integer>& smtrat::BVAnnotation::bounds ( ) const
[inline]
```

0.15.63.2.2 forProduct()

```
static BVAnnotation smtrat::BVAnnotation::forProduct (
    BVAnnotation _factor1,
    BVAnnotation _factor2 ) [inline], [static]
```

0.15.63.2.3 forSum()

```
static BVAnnotation smtrat::BVAnnotation::forSum (
    BVAnnotation _summand1,
    BVAnnotation _summand2 ) [inline], [static]
```

0.15.63.2.4 hasOffset()

```
bool smtrat::BVAnnotation::hasOffset ( ) const [inline]
```

0.15.63.2.5 is_constant()

```
bool smtrat::BVAnnotation::is_constant ( ) const [inline]
```

0.15.63.2.6 isSigned()

```
bool smtrat::BVAnnotation::isSigned ( ) const [inline]
```

0.15.63.2.7 lower_bound()

```
const Integer& smtrat::BVAnnotation::lower_bound ( ) const [inline]
```

0.15.63.2.8 offset()

```
const Integer& smtrat::BVAnnotation::offset ( ) const [inline]
```

0.15.63.2.9 upper_bound()

```
const Integer& smtrat::BVAnnotation::upper_bound ( ) const [inline]
```

0.15.63.2.10 width()

```
std::size_t smtrat::BVAnnotation::width ( ) const [inline]
```

0.15.63.2.11 withOffset() `BVAnnotation smtrat::BVAnnotation::withOffset ( Integer _newOffset ) const [inline]`

0.15.63.2.12 withWidth() `BVAnnotation smtrat::BVAnnotation::withWidth ( std::size_t _width ) const [inline]`

0.15.63.3 Friends And Related Function Documentation

0.15.63.3.1 operator<< `std::ostream& operator<< ( std::ostream & _out, const BVAnnotation & _type ) [friend]`

0.15.64 smtrat::BVDirectEncoder Class Reference

`#include <BVDirectEncoder.h>`

Public Member Functions

- const FormulaSetT & encode (const FormulaT &_inputFormula)
- const std::set< carl::Variable > & introducedVariables () const
- const carl::FastMap< BitVec, Variables > bitvectorBlastings () const
- BVDirectEncoder ()
- ~BVDirectEncoder ()

0.15.64.1 Constructor & Destructor Documentation

0.15.64.1.1 BVDirectEncoder() `smtrat::BVDirectEncoder::BVDirectEncoder ( ) [inline]`

0.15.64.1.2 ~BVDirectEncoder() `smtrat::BVDirectEncoder::~BVDirectEncoder ( ) [inline]`

0.15.64.2 Member Function Documentation

0.15.64.2.1 bitvectorBlastings() `const carl::FastMap<BitVec, Variables> smtrat::BVDirectEncoder↩ ::bitvectorBlastings ( ) const [inline]`

0.15.64.2.2 encode() `const FormulaSetT& smtrat::BVDirectEncoder::encode ( const FormulaT & _inputFormula ) [inline]`

0.15.64.2.3 introducedVariables() `const std::set<carl::Variable>& smtrat::BVDirectEncoder↩ ::introducedVariables ( ) const [inline]`

0.15.65 smtrat::BVModule< Settings > Class Template Reference

`#include <BVModule.h>`

Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- BVModule (const ModuleInput ∗_formula, Conditionals &_conditionals, Manager ∗_manager=NULL)
- ∼BVModule ()
- bool informCore (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void init ()

  Informs all backends about the so far encountered constraints, which have not yet been communicated.
- bool addCore (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- void removeCore (ModuleInput::const_iterator _subformula)

  Removes the subformula of the received formula at the given position to the considered ones of this module.
- void updateModel () const

  Updates the current assignment into the model.
- Answer checkCore ()

  Checks the received formula for consistency.
- bool inform (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void deinform (const FormulaT &_constraint)

  The inverse of informing about a constraint.
- bool add (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- virtual Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_↩ VARIABLE)

  Checks the received formula for consistency.
- virtual void remove (ModuleInput::const_iterator _subformula)

  Removes everything related to the given sub-formula of the received formula.
- virtual void updateAllModels ()

  Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
- virtual std::list< std::vector< carl::Variable > > getModelEqualities () const

  Partition the variables from the current model into equivalence classes according to their assigned value.
- unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const
- virtual unsigned currentlySatisfied (const FormulaT &) const
- bool receivedVariable (carl::Variable::Arg _var) const
- Answer solverState () const
- std::size_t id () const
- void setId (std::size_t _id)

  Sets this modules unique ID to identify itself.
- thread_priority threadPriority () const
- void setThreadPriority (thread_priority _threadPriority)

  Sets the priority of this module to get a thread for running its check procedure.
- const ModuleInput ∗ pReceivedFormula () const
- const ModuleInput & rReceivedFormula () const
- const ModuleInput ∗ pPassedFormula () const
- const ModuleInput & rPassedFormula () const
- const Model & model () const
- const std::vector< Model > & allModels () const

- const std::vector< FormulaSetT > & infeasibleSubsets () const
- const std::vector< Module * > & usedBackends () const
- const carl::FastSet< FormulaT > & constraintsToInform () const
- const carl::FastSet< FormulaT > & informedConstraints () const
- void addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))

  Stores a lemma being a valid formula.
- bool hasLemmas ()

  Checks whether this module or any of its backends provides any lemmas.
- void clearLemmas ()

  Deletes all yet found lemmas.
- const std::vector< Lemma > & lemmas () const
- ModuleInput::const_iterator firstUncheckedReceivedSubformula () const
- ModuleInput::const_iterator firstSubformulaToPass () const
- void receivedFormulaChecked ()

  Notifies that the received formulas has been checked.
- const smtrat::Conditionals & answerFound () const
- bool isPreprocessor () const
- carl::Variable objective () const
- bool is_minimizing () const
- void excludeNotReceivedVariablesFromModel () const

  Excludes all variables from the current model, which do not occur in the received formula.
- void updateLemmas ()

  Stores all lemmas of any backend of this module in its own lemma vector.
- void collectTheoryPropagations ()
- void collectOrigins (const FormulaT &_formula, FormulasT &_origins) const

  Collects the formulas in the given formula, which are part of the received formula.
- void collectOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- bool 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 bool informCore ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.
- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)
- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const

- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

  Copies the infeasible subsets of the passed formula.
- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

Adds a formula to the InformationRelevantFormula.

- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.65.1 Member Typedef Documentation

0.15.65.1.1 SettingsType `template<typename Settings >`

```
typedef Settings smtrat::BVModule< Settings >::SettingsType
```

0.15.65.2 Member Enumeration Documentation

0.15.65.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

0.15.65.3 Constructor & Destructor Documentation

0.15.65.3.1 BVModule() `template<typename Settings >`

```
smtrat::BVModule< Settings >::BVModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = NULL )
```

0.15.65.3.2 ~BVModule() `template<typename Settings >`

```
smtrat::BVModule< Settings >::~BVModule ( )
```

0.15.65.4 Member Function Documentation

0.15.65.4.1 add() `bool smtrat::Module::add (`

```
            ModuleInput::const_iterator _subformula ) [inherited]
```

The module has to take the given sub-formula of the received formula into account.

Parameters

_subformula	The sub-formula to take additionally into account.

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

0.15.65.4.2 addConstraintToInform() `void smtrat::Module::addConstraintToInform (`

```
            const FormulaT & _constraint ) [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

0.15.65.4.3 addCore() [1/2]

```
virtual bool smtrat::Module::addCore (
    [[maybe_unused] ] 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.

0.15.65.4.4 addCore() [2/2]

```
template<typename Settings >
bool smtrat::BVModule< Settings >::addCore (
    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.

0.15.65.4.5 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula
(
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.65.4.6 addLemma()

```
void smtrat::Module::addLemma (
    const FormulaT & _lemma,
    const LemmaType & _lt = LemmaType::NORMAL,
    const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline],
[inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.65.4.7 addOrigin()

```
void smtrat::Module::addOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.65.4.8 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.65.4.9 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.65.4.10 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.65.4.11 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.

0.15.65.4.12 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns

All satisfying assignments, if existent.

0.15.65.4.13 anAnswerFound()

```
bool smtrat::Module::anAnswerFound ( ) const [inline], [protected],
[inherited]
```

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.65.4.14 answerFound()

```
const smtrat::Conditionals& smtrat::Module::answerFound ( ) const
[inline], [inherited]
```

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.65.4.15 backendsModel()

```
const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]
```

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.65.4.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.65.4.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.65.4.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.65.4.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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.

Parameters

_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

0.15.65.4.20 check()

```
Answer smtrat::Module::check (
    bool _final = false,
    bool _full = true,
    carl::Variable _objective = carl::Variable::NO_VARIABLE ) [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.65.4.21 checkCore()

```
template<typename Settings >
Answer smtrat::BVModule< Settings >::checkCore ( ) [virtual]
```

Checks the received formula for consistency.

Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.65.4.22 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.

0.15.65.4.23 checkModel() `unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]`

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.65.4.24 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.65.4.25 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

0.15.65.4.26 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

0.15.65.4.27 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

0.15.65.4.28 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.65.4.29 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

0.15.65.4.30 collectOrigins() [2/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.
If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.65.4.31 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

0.15.65.4.32 constraintsToInform() `const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns

The constraints which the backends must be informed about.

0.15.65.4.33 currentlySatisfied() `virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.65.4.34 currentlySatisfiedByBackend() `unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.65.4.35 deinform() `void smtrat::Module::deinform ( const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.65.4.36 deinformCore() `virtual void smtrat::Module::deinformCore ( [[maybe_unused] ] 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.

0.15.65.4.37 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

0.15.65.4.38 eraseSubformulaFromPassedFormula()

```
ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (
            ModuleInput::iterator _subformula,
            bool _ignoreOrigins = false ) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_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 >.

0.15.65.4.39 evaluateBVFormula()

```
template<typename Settings >
size_t smtrat::BVModule< Settings >::evaluateBVFormula (
            const FormulaT & formula ) [protected]
```

0.15.65.4.40 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

0.15.65.4.41 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
            const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

0.15.65.4.42 firstSubformulaToPass()

```
ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]
```

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.65.4.43 firstUncheckedReceivedSubformula() `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.65.4.44 freeSplittingVariable() `static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.65.4.45 generateTrivialInfeasibleSubset() `void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.65.4.46 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.65.4.47 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.65.4.48 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.65.4.49 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.

0.15.65.4.50 getInformationRelevantFormulas() `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.65.4.51 getModelEqualities()

`std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.65.4.52 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.65.4.53 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.65.4.54 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.

0.15.65.4.55 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.

0.15.65.4.56 hasLemmas() `bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

0.15.65.4.57 hasValidInfeasibleSubset() `bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]`

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.65.4.58 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.65.4.59 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasible↩ Subsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.65.4.60 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.

0.15.65.4.61 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.

0.15.65.4.62 informCore() [1/2]

`virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.65.4.63 informCore() [2/2]

`template<typename Settings > bool smtrat::BVModule< Settings >::informCore ( 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.

0.15.65.4.64 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.

0.15.65.4.65 init()

`template<typename 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.

0.15.65.4.66 is_minimizing()

`bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.65.4.67 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.

0.15.65.4.68 isPreprocessor()

`bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.65.4.69 lemmas()

`const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.65.4.70 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.

0.15.65.4.71 model()

`const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.65.4.72 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.

0.15.65.4.73 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.

0.15.65.4.74 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

0.15.65.4.75 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.65.4.76 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.65.4.77 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.65.4.78 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.65.4.79 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.65.4.80 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.65.4.81 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.

0.15.65.4.82 printInfeasibleSubsets()

`void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.65.4.83 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.

0.15.65.4.84 printPassedFormula()

`void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.65.4.85 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.65.4.86 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.

0.15.65.4.87 receivedFormulaChecked()

```
void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]
```

Notifies that the received formulas has been checked.

0.15.65.4.88 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩InfeasibleSubset (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

0.15.65.4.89 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const [inline], [inherited]
```

0.15.65.4.90 remove()

```
void smtrat::Module::remove (
    ModuleInput::const_iterator _subformula ) [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.
However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

Parameters

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

0.15.65.4.91 removeCore() [1/2]

```
virtual void smtrat::Module::removeCore (
    [[maybe_unused] ] 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.

0.15.65.4.92 removeCore() [2/2]

```
template<typename Settings >
void smtrat::BVModule< Settings >::removeCore (
            ModuleInput::const_iterator _subformula )
```

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.

0.15.65.4.93 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin
(
            ModuleInput::iterator _formula,
            const FormulaT & _origin )  [inline], [protected], [inherited]
```

0.15.65.4.94 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins )  [inline], [protected],
[inherited]
```

0.15.65.4.95 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.65.4.96 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to this module.

0.15.65.4.97 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( )  [inline],
[protected], [virtual], [inherited]
```

Reimplemented in smtrat::PModule.

0.15.65.4.98 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.

0.15.65.4.99 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.

0.15.65.4.100 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.

0.15.65.4.101 solverState()

`Answer smtrat::Module::solverState ( ) const [inline], [inherited]`

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.65.4.102 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 !=.

0.15.65.4.103 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.65.4.104 transferBackendModel() `template<typename Settings > void smtrat::BVModule< Settings >::transferBackendModel ( ) const [protected]`

0.15.65.4.105 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 >.

0.15.65.4.106 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.65.4.107 updateModel() `template<typename 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.

0.15.65.4.108 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).

0.15.65.5 Field Documentation

0.15.65.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.65.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.65.5.3 mBackendsFoundAnswer `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.65.5.4 mConstraintsToInform

`carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.65.5.5 mFinalCheck

`bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.65.5.6 mFirstPosInLastBranches

`std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.65.5.7 mFirstSubformulaToPass

`ModuleInput::iterator smtrat::Module::mFirstSubformulaTo↩Pass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.65.5.8 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.

0.15.65.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.65.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.65.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.65.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.65.5.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std↩::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.65.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.65.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.65.5.16 mNumOfBranchVarsToStore `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.65.5.17 mObjectiveVariable `carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.65.5.18 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

0.15.65.5.19 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.65.5.20 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.65.5.21 mSolverState `std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.65.5.22 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

0.15.65.5.23 mUsedBackends `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.65.5.24 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.66 smtrat::cad::CAD< Settings > Class Template Reference

`#include <CAD.h>`

Public Types

- using SettingsT = Settings

Public Member Functions

- CAD ()
- ~CAD ()
- std::size_t dim () const
- const auto & getVariables () const

- const auto & getProjection () const
- const auto & getLifting () const
- const auto & getConstraints () const
- const auto & getConstraintMap () const
- bool isIdValid (std::size_t id) const
- const auto & getBounds () const
- void reset (const std::vector< carl::Variable > &vars)
- void addConstraint (const ConstraintT &c)
- void removeConstraint (const ConstraintT &c)
- template<typename 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

0.15.66.1 Member Typedef Documentation

0.15.66.1.1 SettingsT `template<typename Settings >`

```
using smtrat::cad::CAD< Settings >::SettingsT = Settings
```

0.15.66.2 Constructor & Destructor Documentation

0.15.66.2.1 CAD() `template<typename Settings >`

```
smtrat::cad::CAD< Settings >::CAD ( ) [inline]
```

0.15.66.2.2 ~CAD() `template<typename Settings >`

```
smtrat::cad::CAD< Settings >::~CAD ( ) [inline]
```

0.15.66.3 Member Function Documentation

0.15.66.3.1 addConstraint() `template<typename Settings >`

```
void smtrat::cad::CAD< Settings >::addConstraint (
            const ConstraintT & c ) [inline]
```

0.15.66.3.2 check()

```
template<typename Settings >
Answer smtrat::cad::CAD< Settings >::check (
            Assignment & assignment,
            std::vector< FormulaSetT > & mis )  [inline]
```

0.15.66.3.3 checkFullSamples()

```
template<typename Settings >
Answer smtrat::cad::CAD< Settings >::checkFullSamples (
            Assignment & assignment )  [inline]
```

0.15.66.3.4 checkPartialSample()

```
template<typename Settings >
template<typename Iterator >
Answer smtrat::cad::CAD< Settings >::checkPartialSample (
            Iterator & it,
            std::size_t level )  [inline]
```

0.15.66.3.5 dim()

```
template<typename Settings >
std::size_t smtrat::cad::CAD< Settings >::dim ( ) const  [inline]
```

0.15.66.3.6 enumerateSolutions()

```
template<typename Settings >
std::vector<Assignment> smtrat::cad::CAD< Settings >::enumerateSolutions ( )  [inline]
```

0.15.66.3.7 evaluateSample()

```
template<typename Settings >
template<typename ConstraintIt >
bool smtrat::cad::CAD< Settings >::evaluateSample (
            Sample & sample,
            const ConstraintIt & constraint,
            Assignment & assignment ) const  [inline]
```

0.15.66.3.8 generateConflictGraph()

```
template<typename Settings >
ConflictGraph smtrat::cad::CAD< Settings >::generateConflictGraph ( ) const  [inline]
```

0.15.66.3.9 generateCovering()

```
template<typename Settings >
auto smtrat::cad::CAD< Settings >::generateCovering ( ) const  [inline]
```

0.15.66.3.10 getBounds()

```
template<typename Settings >
const auto& smtrat::cad::CAD< Settings >::getBounds ( ) const  [inline]
```

0.15.66.3.11 getConstraintMap()

```
template<typename Settings >
const auto& smtrat::cad::CAD< Settings >::getConstraintMap ( ) const  [inline]
```

0.15.66.3.12 getConstraints()

```
template<typename Settings >
const auto& smtrat::cad::CAD< Settings >::getConstraints ( ) const  [inline]
```

0.15.66.3.13 getLifting() template<typename Settings >

```
const auto& smtrat::cad::CAD< Settings >::getLifting ( ) const  [inline]
```

0.15.66.3.14 getProjection() template<typename Settings >

```
const auto& smtrat::cad::CAD< Settings >::getProjection ( ) const  [inline]
```

0.15.66.3.15 getVariables() template<typename Settings >

```
const auto& smtrat::cad::CAD< Settings >::getVariables ( ) const  [inline]
```

0.15.66.3.16 isIdValid() template<typename Settings >

```
bool smtrat::cad::CAD< Settings >::isIdValid (
            std::size_t id ) const  [inline]
```

0.15.66.3.17 removeConstraint() template<typename Settings >

```
void smtrat::cad::CAD< Settings >::removeConstraint (
            const ConstraintT & c )  [inline]
```

0.15.66.3.18 reset() template<typename Settings >

```
void smtrat::cad::CAD< Settings >::reset (
            const std::vector< carl::Variable > & vars )  [inline]
```

0.15.66.4 Friends And Related Function Documentation

0.15.66.4.1 CADCore template<typename Settings >

```
template<CoreHeuristic CH>
friend struct CADCore  [friend]
```

0.15.66.5 Field Documentation

0.15.66.5.1 thp template<typename Settings >

```
debug::TikzHistoryPrinter smtrat::cad::CAD< Settings >::thp
```

0.15.67 smtrat::qe::cad::CAD< Settings > Class Template Reference

```
#include <CAD.h>
```

Public Types

- using 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 ()

0.15.67.1 Member Typedef Documentation

0.15.67.1.1 TreeIterator

```
template<typename Settings >
using smtrat::qe::cad::CAD< Settings >::TreeIterator = typename smtrat::cad::LiftingTree<Settings>↩
::Iterator
```

0.15.67.2 Constructor & Destructor Documentation

0.15.67.2.1 CAD()

```
template<typename Settings >
smtrat::qe::cad::CAD< Settings >::CAD ( ) [inline]
```

0.15.67.3 Member Function Documentation

0.15.67.3.1 addConstraint()

```
template<typename Settings >
void smtrat::qe::cad::CAD< Settings >::addConstraint (
            const ConstraintT & c ) [inline]
```

0.15.67.3.2 addPolynomial()

```
template<typename Settings >
void smtrat::qe::cad::CAD< Settings >::addPolynomial (
            const Poly & p ) [inline]
```

0.15.67.3.3 dim()

```
template<typename Settings >
std::size_t smtrat::qe::cad::CAD< Settings >::dim ( ) const [inline]
```

0.15.67.3.4 getLifting()

```
template<typename Settings >
const auto& smtrat::qe::cad::CAD< Settings >::getLifting ( ) const [inline]
```

0.15.67.3.5 getProjection()

```
template<typename Settings >
const auto& smtrat::qe::cad::CAD< Settings >::getProjection ( ) const [inline]
```

0.15.67.3.6 lift()

```
template<typename Settings >
void smtrat::qe::cad::CAD< Settings >::lift ( ) [inline]
```

0.15.67.3.7 project()

```
template<typename Settings >
void smtrat::qe::cad::CAD< Settings >::project ( ) [inline]
```

0.15.67.3.8 removeConstraint()

```
template<typename Settings >
void smtrat::qe::cad::CAD< Settings >::removeConstraint (
            const ConstraintT & c ) [inline]
```

0.15.67.3.9 removePolynomial()

```
template<typename Settings >
void smtrat::qe::cad::CAD< Settings >::removePolynomial (
            std::size_t level,
            std::size_t id ) [inline]
```

0.15.67.3.10 reset()

```
template<typename Settings >
void smtrat::qe::cad::CAD< Settings >::reset (
            const std::vector< carl::Variable > & vars ) [inline]
```

0.15.68 smtrat::cad::CADConstraints< BT > Class Template Reference

`#include <CADConstraints.h>`

Data Structures

- struct ConstraintComparator

Public Types

- using Callback = std::function< void(const UPoly &, std::size_t, bool)>
- using VariableBounds = vb::VariableBounds< ConstraintT >

Public Member Functions

- CADConstraints (const Callback &onAdd, const Callback &onAddEq, const Callback &onRemove)
- CADConstraints (const CADConstraints &)=delete
- void reset (const std::vector< carl::Variable > &vars)
- const std::vector< carl::Variable > & vars () const
- std::size_t size () const
- bool valid (std::size_t id) const
- const auto & indexed () const
- const auto & ordered () const
- const auto & bounds () const
- const auto & unsatByBounds () const
- std::size_t add (const ConstraintT &c)
- carl::Bitset remove (const ConstraintT &c)

  Removes a constraint.
- const ConstraintT & operator[] (std::size_t id) const
- std::size_t level (std::size_t id) const
- bool checkForTrivialConflict (std::vector< FormulaSetT > &mis) const
- void exportAsDot (std::ostream &out) const

Protected Types

- using ConstraintMap = std::map< ConstraintT, std::size_t, ConstraintComparator >
- using 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)

0.15.68.1 Member Typedef Documentation

0.15.68.1.1 Callback `template<Backtracking BT>`

```
using smtrat::cad::CADConstraints< BT >::Callback = std::function<void(const UPoly&, std↩
::size_t, bool)>
```

0.15.68.1.2 ConstraintIts `template<Backtracking BT>`

```
using smtrat::cad::CADConstraints< BT >::ConstraintIts = std::vector<typename ConstraintMap↩
::iterator>  [protected]
```

0.15.68.1.3 ConstraintMap `template<Backtracking BT>`

```
using smtrat::cad::CADConstraints< BT >::ConstraintMap = std::map<ConstraintT, std::size_t,
ConstraintComparator>  [protected]
```

0.15.68.1.4 VariableBounds `template<Backtracking BT>`

```
using smtrat::cad::CADConstraints< BT >::VariableBounds = vb::VariableBounds<ConstraintT>
```

0.15.68.2 Constructor & Destructor Documentation

0.15.68.2.1 CADConstraints() [1/2] `template<Backtracking BT>`

```
smtrat::cad::CADConstraints< BT >::CADConstraints (
            const Callback & onAdd,
            const Callback & onAddEq,
            const Callback & onRemove )  [inline]
```

0.15.68.2.2 CADConstraints() [2/2] template<Backtracking BT>

```
smtrat::cad::CADConstraints< BT >::CADConstraints (
            const CADConstraints< BT > & ) [delete]
```

0.15.68.3 Member Function Documentation

0.15.68.3.1 add() template<Backtracking BT>

```
std::size_t smtrat::cad::CADConstraints< BT >::add (
            const ConstraintT & c ) [inline]
```

0.15.68.3.2 bounds() template<Backtracking BT>

```
const auto& smtrat::cad::CADConstraints< BT >::bounds ( ) const [inline]
```

0.15.68.3.3 callCallback() template<Backtracking BT>

```
template<typename CB , typename... Args>
void smtrat::cad::CADConstraints< BT >::callCallback (
            const CB & cb,
            const ConstraintT & c,
            Args... args ) const [inline], [protected]
```

0.15.68.3.4 checkForTrivialConflict() template<Backtracking BT>

```
bool smtrat::cad::CADConstraints< BT >::checkForTrivialConflict (
            std::vector< FormulaSetT > & mis ) const [inline]
```

0.15.68.3.5 exportAsDot() template<Backtracking BT>

```
void smtrat::cad::CADConstraints< BT >::exportAsDot (
            std::ostream & out ) const [inline]
```

0.15.68.3.6 indexed() template<Backtracking BT>

```
const auto& smtrat::cad::CADConstraints< BT >::indexed ( ) const [inline]
```

0.15.68.3.7 level() template<Backtracking BT>

```
std::size_t smtrat::cad::CADConstraints< BT >::level (
            std::size_t id ) const [inline]
```

0.15.68.3.8 operator[]() template<Backtracking BT>

```
const ConstraintT& smtrat::cad::CADConstraints< BT >::operator[] (
            std::size_t id ) const [inline]
```

0.15.68.3.9 ordered() template<Backtracking BT>

```
const auto& smtrat::cad::CADConstraints< BT >::ordered ( ) const [inline]
```

0.15.68.3.10 remove()

```
template<Backtracking BT>
carl::Bitset smtrat::cad::CADConstraints< BT >::remove (
            const ConstraintT & c )  [inline]
```

Removes a constraint.

Returns the set of constraint ids that have (possibly) been reassigned and should be cleared from the sample evaluations.

0.15.68.3.11 reset()

```
template<Backtracking BT>
void smtrat::cad::CADConstraints< BT >::reset (
            const std::vector< carl::Variable > & vars )  [inline]
```

0.15.68.3.12 size()

```
template<Backtracking BT>
std::size_t smtrat::cad::CADConstraints< BT >::size ( ) const  [inline]
```

0.15.68.3.13 unsatByBounds()

```
template<Backtracking BT>
const auto& smtrat::cad::CADConstraints< BT >::unsatByBounds ( ) const  [inline]
```

0.15.68.3.14 valid()

```
template<Backtracking BT>
bool smtrat::cad::CADConstraints< BT >::valid (
            std::size_t id ) const  [inline]
```

0.15.68.3.15 vars()

```
template<Backtracking BT>
const std::vector<carl::Variable>& smtrat::cad::CADConstraints< BT >::vars ( ) const  [inline]
```

0.15.68.4 Friends And Related Function Documentation

0.15.68.4.1 operator<<

```
template<Backtracking BT>
template<Backtracking B>
std::ostream& operator<< (
            std::ostream & os,
            const CADConstraints< B > & cc )  [friend]
```

0.15.68.5 Field Documentation

0.15.68.5.1 mActiveConstraintMap

```
template<Backtracking BT>
ConstraintMap smtrat::cad::CADConstraints< BT >::mActiveConstraintMap  [protected]
```

0.15.68.5.2 mAddCallback

```
template<Backtracking BT>
Callback smtrat::cad::CADConstraints< BT >::mAddCallback  [protected]
```

0.15.68.5.3 mAddEqCallback

```
template<Backtracking BT>
Callback smtrat::cad::CADConstraints< BT >::mAddEqCallback  [protected]
```

0.15.68.5.4 mBounds

```
template<Backtracking BT>
VariableBounds smtrat::cad::CADConstraints< BT >::mBounds  [protected]
```

0.15.68.5.5 mConstraintIts template<Backtracking BT>

```
std::vector<typename ConstraintMap::iterator> smtrat::cad::CADConstraints< BT >::mConstraint↩
Its [protected]
```

0.15.68.5.6 mConstraintLevels template<Backtracking BT>

```
std::vector<std::size_t> smtrat::cad::CADConstraints< BT >::mConstraintLevels [protected]
```

0.15.68.5.7 mConstraintMap template<Backtracking BT>

```
ConstraintMap smtrat::cad::CADConstraints< BT >::mConstraintMap [protected]
```

0.15.68.5.8 mIDPool template<Backtracking BT>

```
carl::IDPool smtrat::cad::CADConstraints< BT >::mIDPool [protected]
```

0.15.68.5.9 mRemoveCallback template<Backtracking BT>

```
Callback smtrat::cad::CADConstraints< BT >::mRemoveCallback [protected]
```

0.15.68.5.10 mSatByBounds template<Backtracking BT>

```
carl::Bitset smtrat::cad::CADConstraints< BT >::mSatByBounds [protected]
```

List of constraints that are satisfied by bounds.

0.15.68.5.11 mUnsatByBounds template<Backtracking BT>

```
carl::Bitset smtrat::cad::CADConstraints< BT >::mUnsatByBounds [protected]
```

List of constraints that are infeasible due to bounds.

0.15.68.5.12 mVariables template<Backtracking BT>

```
std::vector<carl::Variable> smtrat::cad::CADConstraints< BT >::mVariables [protected]
```

0.15.69 smtrat::cad::CADCore< CH > Struct Template Reference

```
#include <CADCore.h>
```

0.15.70 smtrat::cad::CADCore< CoreHeuristic::BySample > Struct Reference

```
#include <CADCore.h>
```

Public Member Functions

- template<typename CAD >
  Answer operator() (Assignment &assignment, CAD &cad, bool ignore_nullifications)

0.15.70.1 Member Function Documentation

0.15.70.1.1 operator()() template<typename CAD >

```
Answer smtrat::cad::CADCore< CoreHeuristic::BySample >::operator() (
            Assignment & assignment,
            CAD & cad,
            bool ignore_nullifications ) [inline]
```

0.15.71 smtrat::cad::CADCore< CoreHeuristic::EnumerateAll > Struct Reference

`#include <CADCore.h>`

Public Member Functions

- template<typename CAD >
  Answer operator() (Assignment &assignment, CAD &cad, bool ignore_nullifications)

0.15.71.1 Member Function Documentation

0.15.71.1.1 operator()() template<typename CAD >

```
Answer smtrat::cad::CADCore< CoreHeuristic::EnumerateAll >::operator() (
            Assignment & assignment,
            CAD & cad,
            bool ignore_nullifications ) [inline]
```

0.15.72 smtrat::cad::CADCore< CoreHeuristic::Interleave > Struct Reference

`#include <CADCore.h>`

Public Member Functions

- template<typename It >
  bool preferLifting (const It &it)
- template<typename CAD >
  bool doProjection (CAD &cad)
- template<typename CAD >
  bool doLifting (CAD &cad, bool ignore_nullifications)
- template<typename CAD >
  Answer operator() (Assignment &assignment, CAD &cad, bool ignore_nullifications)

0.15.72.1 Member Function Documentation

0.15.72.1.1 doLifting() template<typename CAD >

```
bool smtrat::cad::CADCore< CoreHeuristic::Interleave >::doLifting (
            CAD & cad,
            bool ignore_nullifications ) [inline]
```

0.15.72.1.2 doProjection() template<typename CAD >

```
bool smtrat::cad::CADCore< CoreHeuristic::Interleave >::doProjection (
            CAD & cad ) [inline]
```

0.15.72.1.3 operator()() template<typename CAD >

```
Answer smtrat::cad::CADCore< CoreHeuristic::Interleave >::operator() (
            Assignment & assignment,
            CAD & cad,
            bool ignore_nullifications ) [inline]
```

0.15.72.1.4 preferLifting() template<typename It >

```
bool smtrat::cad::CADCore< CoreHeuristic::Interleave >::preferLifting (
            const It & it ) [inline]
```

0.15.73 smtrat::cad::CADCore< CoreHeuristic::PreferProjection > Struct Reference

#include <CADCore.h>

Public Member Functions

- template<typename CAD >
  Answer operator() (Assignment &assignment, CAD &cad, bool ignore_nullifications)

0.15.73.1 Member Function Documentation

0.15.73.1.1 operator()() template<typename CAD >

```
Answer smtrat::cad::CADCore< CoreHeuristic::PreferProjection >::operator() (
            Assignment & assignment,
            CAD & cad,
            bool ignore_nullifications ) [inline]
```

0.15.74 smtrat::cad::CADCore< CoreHeuristic::PreferSampling > Struct Reference

#include <CADCore.h>

Public Member Functions

- template<typename CAD >
  Answer operator() (Assignment &assignment, CAD &cad, bool ignore_nullifications)

0.15.74.1 Member Function Documentation

0.15.74.1.1 operator()() template<typename CAD >

```
Answer smtrat::cad::CADCore< CoreHeuristic::PreferSampling >::operator() (
            Assignment & assignment,
            CAD & cad,
            bool ignore_nullifications ) [inline]
```

0.15.75 smtrat::qe::cad::CADElimination Class Reference

#include <CADElimination.h>

Public Member Functions

- CADElimination (const FormulaT &quantifierFreePart, const QEQuery &quantifiers)
- FormulaT eliminateQuantifiers ()

0.15.75.1 Constructor & Destructor Documentation

0.15.75.1.1 CADElimination()

```
smtrat::qe::cad::CADElimination::CADElimination (
            const FormulaT & quantifierFreePart,
            const QEQuery & quantifiers )
```

0.15.75.2 Member Function Documentation

0.15.75.2.1 eliminateQuantifiers() `FormulaT smtrat::qe::cad::CADElimination::eliminateQuantifiers ( )`

0.15.76 smtrat::cad::CADPreprocessor Class Reference

`#include <CADPreprocessor.h>`

Public Member Functions

- CADPreprocessor (const std::vector< carl::Variable > &vars)
- const auto & equalities () const
- const auto & inequalities () const
- void addConstraint (const ConstraintT &c)
- void removeConstraint (const ConstraintT &c)
- bool preprocess ()

  Performs the preprocessing.
- const Model & model () const
- template<typename Map >
  preprocessor::ConstraintUpdate result (const Map &oldC) const
- std::set< FormulaT > getConflict () const
- void postprocessConflict (std::set< FormulaT > &mis) const
- FormulaT simplify (const FormulaT &f) const

Friends

- std::ostream & operator<< (std::ostream &os, const CADPreprocessor &cadpp)

0.15.76.1 Constructor & Destructor Documentation

0.15.76.1.1 CADPreprocessor() `smtrat::cad::CADPreprocessor::CADPreprocessor ( const std::vector< carl::Variable > & vars ) [inline]`

0.15.76.2 Member Function Documentation

0.15.76.2.1 addConstraint() `void smtrat::cad::CADPreprocessor::addConstraint ( const ConstraintT & c )`

0.15.76.2.2 equalities() `const auto& smtrat::cad::CADPreprocessor::equalities ( ) const [inline]`

0.15.76.2.3 getConflict() `std::set< FormulaT > smtrat::cad::CADPreprocessor::getConflict ( ) const`

0.15.76.2.4 inequalities() `const auto& smtrat::cad::CADPreprocessor::inequalities ( ) const [inline]`

0.15.76.2.5 model() `const Model& smtrat::cad::CADPreprocessor::model ( ) const [inline]`

0.15.76.2.6 postprocessConflict() `void smtrat::cad::CADPreprocessor::postprocessConflict ( std::set< FormulaT > & mis ) const`

0.15.76.2.7 preprocess() `bool smtrat::cad::CADPreprocessor::preprocess ( )`

Performs the preprocessing.
Return false if a direct conflict was found, true otherwise.

0.15.76.2.8 removeConstraint() `void smtrat::cad::CADPreprocessor::removeConstraint ( const ConstraintT & c )`

0.15.76.2.9 result() `template<typename Map > preprocessor::ConstraintUpdate smtrat::cad::CADPreprocessor::result ( const Map & oldC ) const [inline]`

0.15.76.2.10 simplify() `FormulaT smtrat::cad::CADPreprocessor::simplify ( const FormulaT & f ) const [inline]`

0.15.76.3 Friends And Related Function Documentation

0.15.76.3.1 operator<< `std::ostream& operator<< ( std::ostream & os, const CADPreprocessor & cadpp ) [friend]`

0.15.77 smtrat::cad::CADPreprocessorSettings Struct Reference

`#include <CADPreprocessor.h>`

Static Public Member Functions

- static void register_settings (SettingsParser &parser)
- static bool register_hook ()

Data Fields

- bool disable_variable_elimination = false
- bool disable_resultants = false

Static Public Attributes

- static const bool dummy = CADPreprocessorSettings::register_hook()

0.15.77.1 Member Function Documentation

0.15.77.1.1 register_hook() `static bool smtrat::cad::CADPreprocessorSettings::register_hook ( ) [inline], [static]`

0.15.77.1.2 register_settings()

`static void smtrat::cad::CADPreprocessorSettings::register_settings ( SettingsParser & parser ) [inline], [static]`

0.15.77.2 Field Documentation

0.15.77.2.1 disable_resultants

`bool smtrat::cad::CADPreprocessorSettings::disable_resultants = false`

0.15.77.2.2 disable_variable_elimination

`bool smtrat::cad::CADPreprocessorSettings::disable_variable_elimination = false`

0.15.77.2.3 dummy

`const bool smtrat::cad::CADPreprocessorSettings::dummy = CADPreprocessorSettings::register_ [static]`

0.15.78 smtrat::qe::cad::CADSettings Struct Reference

`#include <CADElimination.h>`

Static Public Attributes

- static constexpr smtrat::cad::ProjectionType projectionOperator = smtrat::cad::ProjectionType::Brown
- static constexpr smtrat::cad::Incrementality incrementality = smtrat::cad::Incrementality::NONE
- static constexpr smtrat::cad::Backtracking backtracking = smtrat::cad::Backtracking::UNORDERED
- static constexpr CoreHeuristic coreHeuristic = cad::CoreHeuristic::PreferProjection
- static constexpr MISHeuristic misHeuristic = cad::MISHeuristic::GREEDY
- static constexpr std::size_t trivialSampleRadius = 1
- static constexpr bool simplifyProjectionByBounds = true
- static constexpr ProjectionCompareStrategy projectionComparator = cad::ProjectionCompareStrategy::Default
- static constexpr SampleCompareStrategy sampleComparator = cad::SampleCompareStrategy::Default
- static constexpr FullSampleCompareStrategy fullSampleComparator = cad::FullSampleCompareStrategy::Default

0.15.78.1 Field Documentation

0.15.78.1.1 backtracking

`constexpr smtrat::cad::Backtracking smtrat::qe::cad::CADSettings::backtracking = smtrat::cad::Backtracking::UNORDERED [static], [constexpr]`

0.15.78.1.2 coreHeuristic

`constexpr CoreHeuristic smtrat::cad::BaseSettings::coreHeuristic = cad::CoreHeuristic::PreferProjection [static], [constexpr], [inherited]`

0.15.78.1.3 fullSampleComparator

`constexpr FullSampleCompareStrategy smtrat::cad::BaseSettings::fullSampleComparator = cad::FullSampleCompareStrategy::Default [static], [constexpr], [inherited]`

0.15.78.1.4 incrementality

`constexpr smtrat::cad::Incrementality smtrat::qe::cad::CADSettings::incrementality = smtrat::cad::Incrementality::NONE [static], [constexpr]`

0.15.78.1.5 misHeuristic

`constexpr MISHeuristic smtrat::cad::BaseSettings::misHeuristic = cad::MISHeuristic::GF [static], [constexpr], [inherited]`

0.15.78.1.6 projectionComparator

`constexpr ProjectionCompareStrategy smtrat::cad::BaseSettings↩ ::projectionComparator = cad::ProjectionCompareStrategy::Default [static], [constexpr], [inherited]`

0.15.78.1.7 projectionOperator

`constexpr smtrat::cad::ProjectionType smtrat::qe::cad::CADSettings↩ ::projectionOperator = smtrat::cad::ProjectionType::Brown [static], [constexpr]`

0.15.78.1.8 sampleComparator

`constexpr SampleCompareStrategy smtrat::cad::BaseSettings↩ ::sampleComparator = cad::SampleCompareStrategy::Default [static], [constexpr], [inherited]`

0.15.78.1.9 simplifyProjectionByBounds

`constexpr bool smtrat::cad::BaseSettings::simplify↩ ProjectionByBounds = true [static], [constexpr], [inherited]`

0.15.78.1.10 trivialSampleRadius

`constexpr std::size_t smtrat::cad::BaseSettings::trivial↩ SampleRadius = 1 [static], [constexpr], [inherited]`

0.15.79 smtrat::CardinalityEncoder Class Reference

`#include <CardinalityEncoder.h>`

Public Member Functions

- CardinalityEncoder ()
- Rational encodingSize (const ConstraintT &constraint)
- bool canEncode (const ConstraintT &constraint)
- std::string name ()
- std::optional< FormulaT > encode (const ConstraintT &constraint)

  Encodes an arbitrary constraint.

Data Fields

- std::size_t problem_size

Protected Member Functions

- std::optional< FormulaT > doEncode (const ConstraintT &constraint)
- FormulaT generateVarChain (const std::set< carl::Variable > &vars, carl::FormulaType type)

0.15.79.1 Constructor & Destructor Documentation

0.15.79.1.1 CardinalityEncoder()

`smtrat::CardinalityEncoder::CardinalityEncoder ( ) [inline]`

0.15.79.2 Member Function Documentation

0.15.79.2.1 canEncode() `bool smtrat::CardinalityEncoder::canEncode ( const ConstraintT & constraint ) [virtual]`

Implements smtrat::PseudoBoolEncoder.

0.15.79.2.2 doEncode() `std::optional< FormulaT > smtrat::CardinalityEncoder::doEncode ( const ConstraintT & constraint ) [protected], [virtual]`

Implements smtrat::PseudoBoolEncoder.

0.15.79.2.3 encode() `std::optional< FormulaT > smtrat::PseudoBoolEncoder::encode ( const ConstraintT & constraint ) [inherited]`

Encodes an arbitrary constraint.

Returns

encoded formula

0.15.79.2.4 encodingSize() `Rational smtrat::CardinalityEncoder::encodingSize ( const ConstraintT & constraint ) [virtual]`

Reimplemented from smtrat::PseudoBoolEncoder.

0.15.79.2.5 generateVarChain() `FormulaT smtrat::PseudoBoolEncoder::generateVarChain ( const std::set< carl::Variable > & vars, carl::FormulaType type ) [protected], [inherited]`

0.15.79.2.6 name() `std::string smtrat::CardinalityEncoder::name ( ) [inline], [virtual]`

Reimplemented from smtrat::PseudoBoolEncoder.

0.15.79.3 Field Documentation

0.15.79.3.1 problem_size `std::size_t smtrat::PseudoBoolEncoder::problem_size [inherited]`

0.15.80 smtrat::cadcells::representation::cell< H > Struct Template Reference

Note: If connected(i) holds, then the indexed root ordering must contain an ordering between the interval bounds.

`#include <heuristics.h>`

Static Public Member Functions

- template<typename T >
  static std::optional< datastructures::CellRepresentation< T > > compute (datastructures::SampledDerivationRef< T > &der)

0.15.80.1 Detailed Description

template<CellHeuristic H>
struct smtrat::cadcells::representation::cell< H >

Note: If connected(i) holds, then the indexed root ordering must contain an ordering between the interval bounds.

0.15.80.2 Member Function Documentation

0.15.80.2.1 compute()

```
template<CellHeuristic H>
template<typename T >
static std::optional<datastructures::CellRepresentation<T> > smtrat::cadcells::representation::cell<
H >::compute (
            datastructures::SampledDerivationRef< T > & der )  [static]
```

0.15.81 smtrat::cadcells::representation::cell< CellHeuristic::BIGGEST_CELL > Struct Reference

```
#include <heuristics_cell.h>
```

Static Public Member Functions

- template<typename T >
  static std::optional< datastructures::CellRepresentation< T > > compute (datastructures::SampledDerivationRef< T > &der)

0.15.81.1 Member Function Documentation

0.15.81.1.1 compute()

```
template<typename T >
static std::optional<datastructures::CellRepresentation<T> > smtrat::cadcells::representation::cell<
CellHeuristic::BIGGEST_CELL >::compute (
            datastructures::SampledDerivationRef< T > & der )  [inline], [static]
```

0.15.82 smtrat::cadcells::representation::cell< CellHeuristic::BIGGEST_CELL_APPROXIMATION > Struct Reference

```
#include <heuristics_approximation.h>
```

Static Public Member Functions

- template<typename T >
  static std::optional< datastructures::CellRepresentation< T > > compute (datastructures::SampledDerivationRef< T > &der)

0.15.82.1 Member Function Documentation

0.15.82.1.1 compute()

```
template<typename T >
static std::optional<datastructures::CellRepresentation<T> > smtrat::cadcells::representation::cell<
CellHeuristic::BIGGEST_CELL_APPROXIMATION >::compute (
            datastructures::SampledDerivationRef< T > & der )  [inline], [static]
```

0.15.83 smtrat::cadcells::representation::cell< CellHeuristic::BIGGEST_CELL_EW > Struct Reference

```
#include <heuristics_cell.h>
```

Static Public Member Functions

- template<typename T >
  static std::optional< datastructures::CellRepresentation< T > > compute (datastructures::SampledDerivationRef< T > &der)

0.15.83.1 Member Function Documentation

0.15.83.1.1 compute()

```
template<typename T >
static std::optional<datastructures::CellRepresentation<T> > smtrat::cadcells::representation::cell<
CellHeuristic::BIGGEST_CELL_EW >::compute (
            datastructures::SampledDerivationRef< T > & der )  [inline], [static]
```

0.15.84 smtrat::cadcells::representation::cell< CellHeuristic::CHAIN_EQ > Struct Reference

```
#include <heuristics_cell.h>
```

Static Public Member Functions

- template<typename T >
  static std::optional< datastructures::CellRepresentation< T > > compute (datastructures::SampledDerivationRef< T > &der)

0.15.84.1 Member Function Documentation

0.15.84.1.1 compute()

```
template<typename T >
static std::optional<datastructures::CellRepresentation<T> > smtrat::cadcells::representation::cell<
CellHeuristic::CHAIN_EQ >::compute (
            datastructures::SampledDerivationRef< T > & der )  [inline], [static]
```

0.15.85 smtrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS > Struct Reference

```
#include <heuristics_cell.h>
```

Static Public Member Functions

- template<typename T >
  static std::optional< datastructures::CellRepresentation< T > > compute (datastructures::SampledDerivationRef< T > &der)

0.15.85.1 Member Function Documentation

0.15.85.1.1 compute()

```
template<typename T >
static std::optional<datastructures::CellRepresentation<T> > smtrat::cadcells::representation::cell<
CellHeuristic::LOWEST_DEGREE_BARRIERS >::compute (
            datastructures::SampledDerivationRef< T > & der )  [inline], [static]
```

0.15.86 smtrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS_EQ > Struct Reference

```
#include <heuristics_cell.h>
```

Static Public Member Functions

- template<typename T >
  static std::optional< datastructures::CellRepresentation< T > > compute (datastructures::SampledDerivationRef< T > &der)

0.15.86.1 Member Function Documentation

0.15.86.1.1 compute()

```
template<typename T >
static std::optional<datastructures::CellRepresentation<T> > smtrat::cadcells::representation::cell<
CellHeuristic::LOWEST_DEGREE_BARRIERS_EQ >::compute (
            datastructures::SampledDerivationRef< T > & der )  [inline], [static]
```

0.15.87 smtrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS_EW > Struct Reference

```
#include <heuristics_cell.h>
```

Static Public Member Functions

- template<typename T >
  static std::optional< datastructures::CellRepresentation< T > > compute (datastructures::SampledDerivationRef< T > &der)

0.15.87.1 Member Function Documentation

0.15.87.1.1 compute()

```
template<typename T >
static std::optional<datastructures::CellRepresentation<T> > smtrat::cadcells::representation::cell<
CellHeuristic::LOWEST_DEGREE_BARRIERS_EW >::compute (
            datastructures::SampledDerivationRef< T > & der )  [inline], [static]
```

0.15.88 smtrat::cadcells::operators::properties::cell_connected Struct Reference

```
#include <properties.h>
```

Public Member Functions

- std::size_t level () const
- std::size_t hash_on_level () const

Data Fields

- std::size_t lvl

Static Public Attributes

- static constexpr bool is_flag = true

0.15.88.1 Member Function Documentation

0.15.88.1.1 hash_on_level()

```
std::size_t smtrat::cadcells::operators::properties::cell_connected↩
::hash_on_level ( ) const  [inline]
```

0.15.88.1.2 level()

```
std::size_t smtrat::cadcells::operators::properties::cell_connected::level (
) const  [inline]
```

0.15.88.2 Field Documentation

0.15.88.2.1 is_flag `constexpr bool smtrat::cadcells::operators::properties::cell_connected::is_flag = true [static], [constexpr]`

0.15.88.2.2 lvl `std::size_t smtrat::cadcells::operators::properties::cell_connected::lvl`

0.15.89 smtrat::cadcells::representation::approximation::CellApproximator Class Reference

`#include <CellApproximator.h>`

Public Member Functions

- template<typename T >
  CellApproximator (datastructures::SampledDerivationRef< T > &der)
- IR approximate_bound (const IR &p, const RAN &bound, bool below)
- IR approximate_between (const IR &p_l, const IR &p_u, const RAN &l, const RAN &u)
- datastructures::SymbolicInterval compute_cell ()

0.15.89.1 Constructor & Destructor Documentation

0.15.89.1.1 CellApproximator()

```
template<typename T >
smtrat::cadcells::representation::approximation::CellApproximator::CellApproximator (
            datastructures::SampledDerivationRef< T > & der ) [inline]
```

0.15.89.2 Member Function Documentation

0.15.89.2.1 approximate_between()

```
IR smtrat::cadcells::representation::approximation::CellApproximator::approximate_between (
            const IR & p_l,
            const IR & p_u,
            const RAN & l,
            const RAN & u ) [inline]
```

0.15.89.2.2 approximate_bound()

```
IR smtrat::cadcells::representation::approximation::CellApproximator::approximate_bound (
            const IR & p,
            const RAN & bound,
            bool below ) [inline]
```

0.15.89.2.3 compute_cell()

```
datastructures::SymbolicInterval smtrat::cadcells::representation::approximation::CellApproximator::compute_cell ( ) [inline]
```

0.15.90 smtrat::cadcells::datastructures::CellRepresentation< P > Struct Template Reference

Represents a cell.

`#include <representation.h>`

Public Member Functions

- CellRepresentation (SampledDerivationRef< P > &deriv)

Data Fields

- SymbolicInterval description

  Description of a cell.
- IndexedRootOrdering ordering

  An ordering on the roots that protects the cell.
- boost::container::flat_set< PolyRef > equational

  Polynomials that should be projected using the equational constraints projection.
- SampledDerivationRef< P > derivation

  Derivation.

0.15.90.1 Detailed Description

template<typename P>
struct smtrat::cadcells::datastructures::CellRepresentation< P >

Represents a cell.

0.15.90.2 Constructor & Destructor Documentation

0.15.90.2.1 CellRepresentation() `template<typename P >`

`smtrat::cadcells::datastructures::CellRepresentation< P >::CellRepresentation ( SampledDerivationRef< P > & deriv ) [inline]`

0.15.90.3 Field Documentation

0.15.90.3.1 derivation `template<typename P >`

`SampledDerivationRef<P> smtrat::cadcells::datastructures::CellRepresentation< P >::derivation`

Derivation.

0.15.90.3.2 description `template<typename P >`

`SymbolicInterval smtrat::cadcells::datastructures::CellRepresentation< P >::description`

Description of a cell.

0.15.90.3.3 equational `template<typename P >`

`boost::container::flat_set<PolyRef> smtrat::cadcells::datastructures::CellRepresentation< P >::equational`

Polynomials that should be projected using the equational constraints projection.

0.15.90.3.4 ordering `template<typename P >`

`IndexedRootOrdering smtrat::cadcells::datastructures::CellRepresentation< P >::ordering`

An ordering on the roots that protects the cell.

0.15.91 benchmax::slurm::ChunkedSubmitfileProperties Struct Reference

All properties needed to create a submit file.

`#include <SlurmUtilities.h>`

Data Fields

- std::string file_suffix

  Suffix for job and submit file.
- std::string filename_jobs

  Filename of the job list file.
- std::string tmp_dir

  Temporary directory for log files.
- carl::settings::duration limit_time

  Time limit in seconds.
- carl::settings::duration grace_time

  Grace time in seconds.
- carl::settings::binary_quantity limit_memory

  Memory limit in megabytes.
- std::size_t array_size

  Array size.
- std::size_t slice_size

  Slice size.
- std::pair< std::size_t, std::size_t > job_range

  This slice size.

0.15.91.1 Detailed Description

All properties needed to create a submit file.

0.15.91.2 Field Documentation

0.15.91.2.1 array_size `std::size_t benchmax::slurm::ChunkedSubmitfileProperties::array_size`

Array size.

0.15.91.2.2 file_suffix `std::string benchmax::slurm::ChunkedSubmitfileProperties::file_suffix`

Suffix for job and submit file.

0.15.91.2.3 filename_jobs `std::string benchmax::slurm::ChunkedSubmitfileProperties::filename_jobs`

Filename of the job list file.

0.15.91.2.4 grace_time `carl::settings::duration benchmax::slurm::ChunkedSubmitfileProperties::grace_time`

Grace time in seconds.

0.15.91.2.5 job_range `std::pair<std::size_t,std::size_t> benchmax::slurm::ChunkedSubmitfileProperties::job_range`

This slice size.

0.15.91.2.6 limit_memory `carl::settings::binary_quantity benchmax::slurm::ChunkedSubmitfileProperties::limit_memory`

Memory limit in megabytes.

0.15.91.2.7 limit_time `carl::settings::duration benchmax::slurm::ChunkedSubmitfileProperties↩ ::limit_time`

Time limit in seconds.

0.15.91.2.8 slice_size `std::size_t benchmax::slurm::ChunkedSubmitfileProperties::slice_size`

Slice size.

0.15.91.2.9 tmp_dir `std::string benchmax::slurm::ChunkedSubmitfileProperties::tmp_dir`

Temporary directory for log files.

0.15.92 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)

0.15.92.1 Member Function Documentation

0.15.92.1.1 abstraction() `uint32_t Minisat::Clause::abstraction ( ) const [inline]`

0.15.92.1.2 activity() [1/2] `float& Minisat::Clause::activity ( ) [inline]`

0.15.92.1.3 activity() [2/2]

```
float Minisat::Clause::activity ( ) const [inline]
```

0.15.92.1.4 calcAbstraction()

```
void Minisat::Clause::calcAbstraction ( ) [inline]
```

0.15.92.1.5 has_extra()

```
bool Minisat::Clause::has_extra ( ) const [inline]
```

0.15.92.1.6 last()

```
const Lit& Minisat::Clause::last ( ) const [inline]
```

0.15.92.1.7 learnt()

```
bool Minisat::Clause::learnt ( ) const [inline]
```

0.15.92.1.8 mark() [1/2]

```
uint32_t Minisat::Clause::mark ( ) const [inline]
```

0.15.92.1.9 mark() [2/2]

```
void Minisat::Clause::mark (
    uint32_t m ) [inline]
```

0.15.92.1.10 operator const Lit *()

```
Minisat::Clause::operator const Lit * (
    void ) const [inline]
```

0.15.92.1.11 operator[]() [1/2]

```
Lit& Minisat::Clause::operator[] (
    int i ) [inline]
```

0.15.92.1.12 operator[]() [2/2]

```
Lit Minisat::Clause::operator[] (
    int i ) const [inline]
```

0.15.92.1.13 pop()

```
void Minisat::Clause::pop ( ) [inline]
```

0.15.92.1.14 relocate()

```
void Minisat::Clause::relocate (
    CRef c ) [inline]
```

0.15.92.1.15 relocation()

```
CRef Minisat::Clause::relocation ( ) const [inline]
```

0.15.92.1.16 reloced()

```
bool Minisat::Clause::reloced ( ) const [inline]
```

0.15.92.1.17 shrink()

```
void Minisat::Clause::shrink (
    int i ) [inline]
```

0.15.92.1.18 size()

```
int Minisat::Clause::size ( ) const [inline]
```

0.15.92.1.19 strengthen()

```
void Minisat::Clause::strengthen (
    Lit p ) [inline]
```

0.15.92.1.20 subsumes()

```
Lit Minisat::Clause::subsumes (
    const Clause & other ) const [inline]
```

0.15.92.1.21 type()

```
unsigned Minisat::Clause::type ( ) const [inline]
```

0.15.92.2 Friends And Related Function Documentation

0.15.92.2.1 Clause_new

```
template<class V >
Clause* Clause_new (
    const V & ps,
    bool learnt = false ) [friend]
```

0.15.92.2.2 ClauseAllocator

```
friend class ClauseAllocator [friend]
```

0.15.92.3 Field Documentation

0.15.92.3.1 abs

```
uint32_t Minisat::Clause::abs
```

0.15.92.3.2 act

```
float Minisat::Clause::act
```

0.15.92.3.3 has_extra

```
unsigned Minisat::Clause::has_extra
```

0.15.92.3.4 lit

```
Lit Minisat::Clause::lit
```

0.15.92.3.5 mark

```
unsigned Minisat::Clause::mark
```

0.15.92.3.6 rel

```
CRef Minisat::Clause::rel
```

0.15.92.3.7 reloced

```
unsigned Minisat::Clause::reloced
```

0.15.92.3.8 size

```
unsigned Minisat::Clause::size
```

0.15.92.3.9 type

```
unsigned Minisat::Clause::type
```

0.15.93 Minisat::ClauseAllocator Class Reference

```
#include <SolverTypes.h>
```

Public Types

- enum
- enum
- typedef uint32_t Ref

Public Member Functions

- ClauseAllocator (uint32_t start_cap)
- ClauseAllocator ()
- void moveTo (ClauseAllocator &to)
- template<class Lits >
  CRef alloc (const Lits &ps, unsigned _type)
- Clause & operator[ ] (Ref r)
- const Clause & operator[ ] (Ref r) const
- Clause ∗ lea (Ref r)
- const Clause ∗ lea (Ref r) const
- Ref ael (const Clause ∗t)
- void free (CRef cid)
- void reloc (CRef &cr, ClauseAllocator &to)
- uint32_t size () const
- uint32_t wasted () const
- Ref alloc (int size)
- void free (int size)
- Ref ael (const uint32_t ∗t)
- void moveTo (RegionAllocator &to)

Data Fields

- bool extra_clause_field

0.15.93.1 Member Typedef Documentation

0.15.93.1.1 Ref `typedef uint32_t Minisat::RegionAllocator< uint32_t >::Ref [inherited]`

0.15.93.2 Member Enumeration Documentation

0.15.93.2.1 anonymous enum `anonymous enum [inherited]`

0.15.93.2.2 anonymous enum `anonymous enum [inherited]`

0.15.93.3 Constructor & Destructor Documentation

0.15.93.3.1 ClauseAllocator() [1/2] `Minisat::ClauseAllocator::ClauseAllocator ( uint32_t start_cap ) [inline]`

0.15.93.3.2 ClauseAllocator() [2/2] `Minisat::ClauseAllocator::ClauseAllocator ( ) [inline]`

0.15.93.4 Member Function Documentation

0.15.93.4.1 ael() [1/2]

```
Ref Minisat::ClauseAllocator::ael (
    const Clause * t ) [inline]
```

0.15.93.4.2 ael() [2/2]

```
Ref Minisat::RegionAllocator< uint32_t >::ael (
    const uint32_t * t ) [inline], [inherited]
```

0.15.93.4.3 alloc() [1/2]

```
template<class Lits >
CRef Minisat::ClauseAllocator::alloc (
    const Lits & ps,
    unsigned _type ) [inline]
```

0.15.93.4.4 alloc() [2/2]

```
RegionAllocator< uint32_t >::Ref Minisat::RegionAllocator< uint32_↩
t >::alloc (
    int size ) [inherited]
```

0.15.93.4.5 free() [1/2]

```
void Minisat::ClauseAllocator::free (
    CRef cid ) [inline]
```

0.15.93.4.6 free() [2/2]

```
void Minisat::RegionAllocator< uint32_t >::free (
    int size ) [inline], [inherited]
```

0.15.93.4.7 lea() [1/2]

```
Clause* Minisat::ClauseAllocator::lea (
    Ref r ) [inline]
```

0.15.93.4.8 lea() [2/2]

```
const Clause* Minisat::ClauseAllocator::lea (
    Ref r ) const [inline]
```

0.15.93.4.9 moveTo() [1/2]

```
void Minisat::ClauseAllocator::moveTo (
    ClauseAllocator & to ) [inline]
```

0.15.93.4.10 moveTo() [2/2]

```
void Minisat::RegionAllocator< uint32_t >::moveTo (
    RegionAllocator< uint32_t > & to ) [inline], [inherited]
```

0.15.93.4.11 operator[]() [1/2]

```
Clause& Minisat::ClauseAllocator::operator[] (
    Ref r ) [inline]
```

0.15.93.4.12 operator[]() [2/2]

```
const Clause& Minisat::ClauseAllocator::operator[] (
    Ref r ) const [inline]
```

0.15.93.4.13 reloc()

```
void Minisat::ClauseAllocator::reloc (
    CRef & cr,
    ClauseAllocator & to )  [inline]
```

0.15.93.4.14 size()

```
uint32_t Minisat::RegionAllocator< uint32_t >::size (
    void ) const  [inline], [inherited]
```

0.15.93.4.15 wasted()

```
uint32_t Minisat::RegionAllocator< uint32_t >::wasted ( ) const  [inline], [inherited]
```

0.15.93.5 Field Documentation

0.15.93.5.1 extra_clause_field

```
bool Minisat::ClauseAllocator::extra_clause_field
```

0.15.94 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)

0.15.94.1 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).

0.15.94.2 Member Typedef Documentation

0.15.94.2.1 const_iterator

```
using smtrat::mcsat::ClauseChain::const_iterator = typename std↩
::vector<Link>::const_iterator
```

0.15.94.3 Constructor & Destructor Documentation

0.15.94.3.1 ClauseChain()

```
smtrat::mcsat::ClauseChain::ClauseChain ( ) [inline]
```

0.15.94.4 Member Function Documentation

0.15.94.4.1 appendConflicting()

```
void smtrat::mcsat::ClauseChain::appendConflicting (
    const FormulaT && clause ) [inline]
```

Append a conflicting clause (regarding the current assignment).

0.15.94.4.2 appendOptional()

```
void smtrat::mcsat::ClauseChain::appendOptional (
    const FormulaT && clause ) [inline]
```

Append an additional clause which is neither propagating nor conflicting.
Can be used to pass additional knowledge.

0.15.94.4.3 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.

0.15.94.4.4 begin()

```
const_iterator smtrat::mcsat::ClauseChain::begin ( ) const [inline]
```

Returns

A constant iterator to the beginning of this chain.

0.15.94.4.5 chain()

```
const std::vector<Link>& smtrat::mcsat::ClauseChain::chain ( ) const [inline]
```

Returns

A vector representing this chain.

0.15.94.4.6 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

0.15.94.4.7 end()

```
const_iterator smtrat::mcsat::ClauseChain::end ( ) const [inline]
```

Returns

A constant iterator to the end of this chain.

0.15.94.4.8 from_formula()

```
ClauseChain smtrat::mcsat::ClauseChain::from_formula (
    const FormulaT & f,
    const Model & model,
    bool with_equivalence ) [static]
```

Transforms a given Boolean conjunction over AND and OR to CNF via Tseitin-Transformation so that, if the input formula is conflicting under the current assignment, after all clauses in "implications" have been propagated in the given order, the returned formula evaluates to false.

0.15.94.4.9 resolve()

```
FormulaT smtrat::mcsat::ClauseChain::resolve ( ) const
```

Performs resolution on the chain.
Uses the last clause for resolution.

Returns

A single clause conflicting under the current assignment.

0.15.94.4.10 to_formula()

```
FormulaT smtrat::mcsat::ClauseChain::to_formula ( ) const
```

Transforms the clause chain into a formula (containing Tseitin variables).

0.15.94.5 Friends And Related Function Documentation

0.15.94.5.1 operator<<

```
std::ostream& operator<< (
    std::ostream & stream,
    const ClauseChain & chain ) [friend]
```

0.15.95 smtrat::sat::detail::ClauseChecker Struct Reference

```
#include <ClauseChecker.h>
```

Public Member Functions

- Model buildModel () const
- void operator() (const FormulaT &formula) const
- void operator() (const FormulasT &formulas) const
- template<typename VM , typename BCM >
  void operator() (const Minisat::Clause &c, const VM &vm, const BCM &bcm) const

0.15.95.1 Member Function Documentation

0.15.95.1.1 buildModel() `Model smtrat::sat::detail::ClauseChecker::buildModel ( ) const [inline]`

0.15.95.1.2 operator()() [1/3]

```
void smtrat::sat::detail::ClauseChecker::operator() (
    const FormulasT & formulas ) const [inline]
```

0.15.95.1.3 operator()() [2/3]

```
void smtrat::sat::detail::ClauseChecker::operator() (
    const FormulaT & formula ) const [inline]
```

0.15.95.1.4 operator()() [3/3]

```
template<typename VM , typename BCM >
void smtrat::sat::detail::ClauseChecker::operator() (
    const Minisat::Clause & c,
    const VM & vm,
    const BCM & bcm ) const [inline]
```

0.15.96 smtrat::CMakeOptionPrinter Struct Reference

`#include <compile_information.h>`

Data Fields

- bool advanced

0.15.96.1 Field Documentation

0.15.96.1.1 advanced `bool smtrat::CMakeOptionPrinter::advanced`

0.15.97 Minisat::CMap< T > Class Template Reference

`#include <SolverTypes.h>`

Public Member Functions

- void clear ()
- int size () const
- void insert (CRef cr, const T &t)
- void growTo (CRef cr, const T &t)
- void remove (CRef cr)
- bool has (CRef cr, T &t)
- const T & operator[ ] (CRef cr) const
- T & operator[ ] (CRef cr)
- int bucket_count () const
- const vec< typename HashTable::Pair > & bucket (int i) const
- void moveTo (CMap &other)
- void debug ()

0.15.97.1 Member Function Documentation

0.15.97.1.1 bucket()

```
template<class T >
const vec<typename HashTable::Pair>& Minisat::CMap< T >::bucket (
    int i ) const [inline]
```

0.15.97.1.2 bucket_count() template<class T >

```
int Minisat::CMap< T >::bucket_count ( ) const  [inline]
```

0.15.97.1.3 clear() template<class T >

```
void Minisat::CMap< T >::clear ( )  [inline]
```

0.15.97.1.4 debug() template<class T >

```
void Minisat::CMap< T >::debug ( )  [inline]
```

0.15.97.1.5 growTo() template<class T >

```
void Minisat::CMap< T >::growTo (
            CRef cr,
            const T & t )  [inline]
```

0.15.97.1.6 has() template<class T >

```
bool Minisat::CMap< T >::has (
            CRef cr,
            T & t )  [inline]
```

0.15.97.1.7 insert() template<class T >

```
void Minisat::CMap< T >::insert (
            CRef cr,
            const T & t )  [inline]
```

0.15.97.1.8 moveTo() template<class T >

```
void Minisat::CMap< T >::moveTo (
            CMap< T > & other )  [inline]
```

0.15.97.1.9 operator[]() [1/2] template<class T >

```
T& Minisat::CMap< T >::operator[] (
            CRef cr )  [inline]
```

0.15.97.1.10 operator[]() [2/2] template<class T >

```
const T& Minisat::CMap< T >::operator[] (
            CRef cr ) const  [inline]
```

0.15.97.1.11 remove() template<class T >

```
void Minisat::CMap< T >::remove (
            CRef cr )  [inline]
```

0.15.97.1.12 size() template<class T >

```
int Minisat::CMap< T >::size ( ) const  [inline]
```

0.15.98 smtrat::CNFerModule Class Reference

```
#include <CNFerModule.h>
```

Data Structures

- struct SettingsType

Public Types

- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- CNFerModule (const ModuleInput ∗, Conditionals &, Manager ∗const =NULL)

  Constructs a CNFerModule.
- ∼CNFerModule ()

  Destructs a CNFerModule.
- Answer checkCore ()

  Checks the received formula for consistency.
- bool isPreprocessor () const
- bool appliedPreprocessing () const
- bool add (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- void remove (ModuleInput::const_iterator _subformula)

  Removes everything related to the given sub-formula of the received formula.
- Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)

  Checks the received formula for consistency.
- Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- std::pair< bool, FormulaT > getReceivedFormulaSimplified ()
- void updateModel () const

  Updates the current assignment into the model.
- bool inform (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void deinform (const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual void init ()

  Informs all backends about the so far encountered constraints, which have not yet been communicated.
- virtual void updateAllModels ()

  Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
- virtual std::list< std::vector< carl::Variable > > getModelEqualities () const

  Partition the variables from the current model into equivalence classes according to their assigned value.
- unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const
- virtual unsigned currentlySatisfied (const FormulaT &) const
- bool receivedVariable (carl::Variable::Arg _var) const
- Answer solverState () const
- std::size_t id () const
- void setId (std::size_t _id)

  Sets this modules unique ID to identify itself.
- thread_priority threadPriority () const
- void setThreadPriority (thread_priority _threadPriority)

  Sets the priority of this module to get a thread for running its check procedure.
- const ModuleInput ∗ pReceivedFormula () const
- const ModuleInput & rReceivedFormula () const
- const ModuleInput ∗ pPassedFormula () const

- const ModuleInput & rPassedFormula () const
- const Model & model () const
- const std::vector< Model > & allModels () const
- const std::vector< FormulaSetT > & infeasibleSubsets () const
- const std::vector< Module * > & usedBackends () const
- const carl::FastSet< FormulaT > & constraintsToInform () const
- const carl::FastSet< FormulaT > & informedConstraints () const
- void addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))

  Stores a lemma being a valid formula.
- bool hasLemmas ()

  Checks whether this module or any of its backends provides any lemmas.
- void clearLemmas ()

  Deletes all yet found lemmas.
- const std::vector< Lemma > & lemmas () const
- ModuleInput::const_iterator firstUncheckedReceivedSubformula () const
- ModuleInput::const_iterator firstSubformulaToPass () const
- void receivedFormulaChecked ()

  Notifies that the received formulas has been checked.
- const 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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.
- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)
- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.

- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

  Copies the infeasible subsets of the passed formula.
- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.98.1 Member Enumeration Documentation

0.15.98.1.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

0.15.98.2 Constructor & Destructor Documentation

0.15.98.2.1 CNFerModule()

```
smtrat::CNFerModule::CNFerModule (
    const ModuleInput * ,
    Conditionals & ,
    Manager * const  = NULL )
```

Constructs a CNFerModule.

0.15.98.2.2 ~CNFerModule()

```
smtrat::CNFerModule::~CNFerModule ( )
```

Destructs a CNFerModule.

0.15.98.3 Member Function Documentation

0.15.98.3.1 add()

```
bool smtrat::PModule::add (
    ModuleInput::const_iterator _subformula )  [inherited]
```

The module has to take the given sub-formula of the received formula into account.

Parameters

_subformula	The sub-formula to take additionally into account.

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

0.15.98.3.2 addConstraintToInform()

```
void smtrat::Module::addConstraintToInform (
    const FormulaT & _constraint )  [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

0.15.98.3.3 addCore()

```
virtual bool smtrat::Module::addCore (
    [[maybe_unused] ] 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.

0.15.98.3.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.98.3.5 addLemma()

```
void smtrat::Module::addLemma (
    const FormulaT & _lemma,
    const LemmaType & _lt = LemmaType::NORMAL,
    const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline],
[inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.98.3.6 addOrigin()

```
void smtrat::Module::addOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.98.3.7 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.98.3.8 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula )  [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.98.3.9 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.

0.15.98.3.10 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.

0.15.98.3.11 allModels()

`const std::vector<Model>& smtrat::Module::allModels ( ) const [inline], [inherited]`

Returns

All satisfying assignments, if existent.

0.15.98.3.12 anAnswerFound()

`bool smtrat::Module::anAnswerFound ( ) const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.98.3.13 answerFound()

`const smtrat::Conditionals& smtrat::Module::answerFound ( ) const [inline], [inherited]`

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.98.3.14 appliedPreprocessing()

`bool smtrat::PModule::appliedPreprocessing ( ) const [inline], [inherited]`

Returns

true, if the current received formula has been simplified and the result of this simplification is stored in the passed formula.

0.15.98.3.15 backendsModel()

`const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.98.3.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.98.3.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.98.3.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline],
[protected], [inherited]
```

0.15.98.3.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.98.3.20 check()

```
Answer smtrat::PModule::check (
    bool _final = false,
    bool _full = true,
    carl::Variable _objective = carl::Variable::NO_VARIABLE ) [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.98.3.21 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.

0.15.98.3.22 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.

0.15.98.3.23 checkModel()

`unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]`

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.98.3.24 cleanModel()

`void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.98.3.25 clearLemmas()

`void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

0.15.98.3.26 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

0.15.98.3.27 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

0.15.98.3.28 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.98.3.29 collectOrigins() [1/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

0.15.98.3.30 collectOrigins() [2/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inherited]`

Collects the formulas in the given formula, which are part of the received formula.
If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.98.3.31 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

0.15.98.3.32 constraintsToInform() `const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns

The constraints which the backends must be informed about.

0.15.98.3.33 currentlySatisfied() `virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.98.3.34 currentlySatisfiedByBackend()

```
unsigned smtrat::Module::currentlySatisfiedByBackend (
    const FormulaT & _formula ) const [inherited]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.98.3.35 deinform()

```
void smtrat::Module::deinform (
    const FormulaT & _constraint ) [inherited]
```

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.98.3.36 deinformCore()

```
virtual void smtrat::Module::deinformCore (
    [[maybe_unused] ] 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.

0.15.98.3.37 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

0.15.98.3.38 eraseSubformulaFromPassedFormula()

```
ModuleInput::iterator smtrat::Module::erase↩
SubformulaFromPassedFormula (
    ModuleInput::iterator _subformula,
    bool _ignoreOrigins = false ) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_subformula	The sub-formula to remove from the passed formula.

Parameters

_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 >.

0.15.98.3.39 excludeNotReceivedVariablesFromModel()

`void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

0.15.98.3.40 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
            const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

0.15.98.3.41 firstSubformulaToPass()

`ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.98.3.42 firstUncheckedReceivedSubformula()

`ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.98.3.43 freeSplittingVariable()

```
static void smtrat::Module::freeSplittingVariable (
            const FormulaT & _splittingVariable ) [inline], [static], [inherited]
```

0.15.98.3.44 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.98.3.45 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.98.3.46 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.98.3.47 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.98.3.48 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.

0.15.98.3.49 getInformationRelevantFormulas() `const std::set< FormulaT > & smtrat::Module↩ ::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.98.3.50 getModelEqualities() `std::list< std::vector< carl::Variable > > smtrat::Module↩ ::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.98.3.51 getOrigins() [1/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.98.3.52 getOrigins() [2/3]

`void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.98.3.53 getOrigins() [3/3]

`const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]`

Gets the origins of the passed formula at the given position.

Parameters

_formula	The position of a formula in the passed formulas.

Returns

The origins of the passed formula at the given position.

0.15.98.3.54 getReceivedFormulaSimplified()

`std::pair< bool, FormulaT > smtrat::PModule::getReceivedFormulaSimplified ( ) [virtual], [inherited]`

Returns

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's reveived formula, if the Boolean is true.

Reimplemented from smtrat::Module.

0.15.98.3.55 hasLemmas()

`bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

0.15.98.3.56 hasValidInfeasibleSubset()

`bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]`

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.98.3.57 id()

`std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.98.3.58 infeasibleSubsets()

`const std::vector<FormulaSetT>& smtrat::Module::infeasible↩ Subsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.98.3.59 inform()

`bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.
It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

Parameters

_constraint	The constraint to inform about.

Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

0.15.98.3.60 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.

0.15.98.3.61 informCore()

`virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.98.3.62 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.

0.15.98.3.63 init() `void smtrat::Module::init ( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not be called twice and only before the first runBackends call.
Reimplemented in smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntBlastModule< Settings >, smtrat::FPPModule< Settings >, smtrat::CurryModule< Settings >, and smtrat::BVModule< Settings >.

0.15.98.3.64 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.98.3.65 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.

0.15.98.3.66 isPreprocessor() `bool smtrat::PModule::isPreprocessor ( ) const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.98.3.67 lemmas() `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.98.3.68 merge() `std::vector< FormulaT > smtrat::Module::merge ( const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.
({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

Parameters

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns

The vector being the two given vectors merged.

0.15.98.3.69 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.98.3.70 modelsDisjoint()

`bool smtrat::Module::modelsDisjoint ( const Model & _modelA, const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

Parameters

_modelA	The first model to check for.
_modelB	The second model to check for.

Returns

true, if there is no variable assigned by both models; false, otherwise.

0.15.98.3.71 moduleName()

`virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]`

Returns

The name of the given module type as name.

Reimplemented in smtrat::VSModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::STropModule< Settings >, smtrat::SplitSOSModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntEqModule< Settings >, smtrat::IntBlastModule< Settings >, smtrat::IncWidthModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::GBModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::CurryModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::BVModule< Settings >, and smtrat::BEModule< Settings >.

0.15.98.3.72 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

0.15.98.3.73 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.98.3.74 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.98.3.75 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.98.3.76 pPassedFormula() `const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.98.3.77 pReceivedFormula() `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.98.3.78 print() `void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.98.3.79 printAllModels() `void smtrat::Module::printAllModels ( std::ostream & _out = std::cout ) [inherited]`

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.98.3.80 printInfeasibleSubsets() `void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.98.3.81 printModel() `void smtrat::Module::printModel ( std::ostream & _out = std::cout ) const [inherited]`

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.98.3.82 printPassedFormula()

`void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.98.3.83 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.98.3.84 probablyLooping()

`bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.98.3.85 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.98.3.86 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

0.15.98.3.87 receivedVariable()

`bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

0.15.98.3.88 remove()

`void smtrat::PModule::remove ( ModuleInput::const_iterator _subformula ) [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.
However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

Parameters

_subformula	The sub formula of the received formula to remove.

Reimplemented from smtrat::Module.

0.15.98.3.89 removeCore()

```
virtual void smtrat::Module::removeCore (
    [[maybe_unused] ] 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.

0.15.98.3.90 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin
(
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

0.15.98.3.91 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

0.15.98.3.92 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.98.3.93 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to this module.

0.15.98.3.94 runBackends() [1/2]

```
virtual Answer smtrat::PModule::runBackends ( ) [inline],
[virtual], [inherited]
```

Reimplemented from smtrat::Module.

0.15.98.3.95 runBackends() [2/2]

```
Answer smtrat::PModule::runBackends (
    bool _final,
    bool _full,
    carl::Variable _objective ) [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

Parameters

_final	true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT.
_full	false, if this module should avoid too expensive procedures and rather return unknown instead.
_objective	if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good.

Returns

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.98.3.96 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.

0.15.98.3.97 setThreadPriority()

```
void smtrat::Module::setThreadPriority (
    thread_priority _threadPriority ) [inline], [inherited]
```

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.98.3.98 solverState()

```
Answer smtrat::Module::solverState ( ) const [inline], [inherited]
```

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.98.3.99 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 !=.

0.15.98.3.100 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.98.3.101 updateAllModels() `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.98.3.102 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.98.3.103 updateModel() `void smtrat::PModule::updateModel ( ) const [virtual], [inherited]`

Updates the current assignment into the model.
Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.
Reimplemented from smtrat::Module.

0.15.98.3.104 usedBackends() `const std::vector<Module*>& smtrat::Module::usedBackends ( ) const [inline], [inherited]`

Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.98.4 Field Documentation

0.15.98.4.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.98.4.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.98.4.3 mBackendsFoundAnswer `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.98.4.4 mConstraintsToInform

`carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.98.4.5 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

0.15.98.4.6 mFirstPosInLastBranches

`std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.98.4.7 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.

0.15.98.4.8 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.

0.15.98.4.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.98.4.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.98.4.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.98.4.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.98.4.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std↩::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.98.4.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.98.4.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.98.4.16 mModelComputed `bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.98.4.17 mNumOfBranchVarsToStore `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.98.4.18 mObjectiveVariable `carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.98.4.19 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

0.15.98.4.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.98.4.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.98.4.22 mSolverState `std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.98.4.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

0.15.98.4.24 mUsedBackends `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.98.4.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.99 smtrat::CoCoAGBModule< Settings > Class Template Reference

`#include <CoCoAGBModule.h>`

Public Types

- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- bool inform (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void deinform (const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual void init ()

  Informs all backends about the so far encountered constraints, which have not yet been communicated.
- bool add (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- virtual Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_↩ VARIABLE)

  Checks the received formula for consistency.
- virtual void remove (ModuleInput::const_iterator _subformula)

  Removes everything related to the given sub-formula of the received formula.
- virtual void updateModel () const

  Updates the model, if the solver has detected the consistency of the received formula, beforehand.
- virtual void updateAllModels ()

  Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
- virtual std::list< std::vector< carl::Variable > > getModelEqualities () const

  Partition the variables from the current model into equivalence classes according to their assigned value.
- unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const
- virtual unsigned currentlySatisfied (const FormulaT &) const
- bool receivedVariable (carl::Variable::Arg _var) const
- Answer solverState () const
- std::size_t id () const
- void setId (std::size_t _id)

  Sets this modules unique ID to identify itself.
- thread_priority threadPriority () const
- void setThreadPriority (thread_priority _threadPriority)

  Sets the priority of this module to get a thread for running its check procedure.
- const ModuleInput * pReceivedFormula () const
- const ModuleInput & rReceivedFormula () const
- const ModuleInput * pPassedFormula () const
- const ModuleInput & rPassedFormula () const
- const Model & model () const
- const std::vector< Model > & allModels () const
- const std::vector< FormulaSetT > & infeasibleSubsets () const
- const std::vector< Module * > & usedBackends () const
- const carl::FastSet< FormulaT > & constraintsToInform () const
- const carl::FastSet< FormulaT > & informedConstraints () const
- void addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))

  Stores a lemma being a valid formula.
- bool hasLemmas ()

  Checks whether this module or any of its backends provides any lemmas.
- void clearLemmas ()

  Deletes all yet found lemmas.
- const std::vector< Lemma > & lemmas () const
- ModuleInput::const_iterator firstUncheckedReceivedSubformula () const
- ModuleInput::const_iterator firstSubformulaToPass () const
- void receivedFormulaChecked ()

  Notifies that the received formulas has been checked.

- const 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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] 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 ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.
- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)
- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.

- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

  Copies the infeasible subsets of the passed formula.
- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.99.1 Member Enumeration Documentation

0.15.99.1.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

0.15.99.2 Member Function Documentation

0.15.99.2.1 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.

0.15.99.2.2 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 >.

0.15.99.2.3 addCore()

`virtual bool smtrat::Module::addCore ( [[maybe_unused] ] 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.

0.15.99.2.4 addInformationRelevantFormula()

`void smtrat::Module::addInformationRelevantFormula ( const FormulaT & formula ) [protected], [inherited]`

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.99.2.5 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.

0.15.99.2.6 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.

0.15.99.2.7 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.

0.15.99.2.8 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.

0.15.99.2.9 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.

0.15.99.2.10 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.

0.15.99.2.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const  [inline],
[inherited]
```

Returns

All satisfying assignments, if existent.

0.15.99.2.12 anAnswerFound() `bool smtrat::Module::anAnswerFound ( ) const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.99.2.13 answerFound() `const smtrat::Conditionals& smtrat::Module::answerFound ( ) const [inline], [inherited]`

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.99.2.14 backendsModel() `const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.99.2.15 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.99.2.16 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.99.2.17 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.99.2.18 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
```

```
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false )  [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.99.2.19 check()

```
Answer smtrat::Module::check (
bool _final = false,
bool _full = true,
carl::Variable _objective = carl::Variable::NO_VARIABLE )  [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.99.2.20 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::VSModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::SymmetryModule< Settings >, smtrat::STropModule< Settings >, smtrat::SplitSOSModule< Settings >, smtrat::SATModule< Settings >, smtrat::PFEModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Se smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::MCBModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASetti smtrat::LRAModule< LRASettings1 >, smtrat::IntEqModule< Settings >, smtrat::IntBlastModule< Settings >, smtrat::IncWidthModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::ICEModule< Settings >, smtrat::GBPPModule< Settings >, smtrat::GBModule< Settings >, smtrat::FPPModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::ESModule< Settings smtrat::EMModule< Settings >, smtrat::CurryModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::CoveringNGModule< Settings >, smtrat::CNFerModule, smtrat::BVModule< Settings >, and smtrat::BEModule< Settings >.

0.15.99.2.21 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.

0.15.99.2.22 checkModel()

```
unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]
```

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.99.2.23 cleanModel()

```
void smtrat::Module::cleanModel ( ) const [inline], [protected],
[inherited]
```

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.99.2.24 clearLemmas()

```
void smtrat::Module::clearLemmas ( ) [inline], [inherited]
```

Deletes all yet found lemmas.

0.15.99.2.25 clearModel()

```
void smtrat::Module::clearModel ( ) const [inline], [protected],
[inherited]
```

Clears the assignment, if any was found.

0.15.99.2.26 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

0.15.99.2.27 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.99.2.28 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

0.15.99.2.29 collectOrigins() [2/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.
If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.99.2.30 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

0.15.99.2.31 constraintsToInform() `const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns

The constraints which the backends must be informed about.

0.15.99.2.32 currentlySatisfied()

```
virtual unsigned smtrat::Module::currentlySatisfied (
    const FormulaT & ) const [inline], [virtual], [inherited]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.99.2.33 currentlySatisfiedByBackend()

```
unsigned smtrat::Module::currentlySatisfiedByBackend
(
    const FormulaT & _formula ) const [inherited]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.99.2.34 deinform()

```
void smtrat::Module::deinform (
    const FormulaT & _constraint ) [inherited]
```

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.99.2.35 deinformCore()

```
virtual void smtrat::Module::deinformCore (
    [[maybe_unused] ] 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.

0.15.99.2.36 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

0.15.99.2.37 eraseSubformulaFromPassedFormula()

```
ModuleInput::iterator smtrat::Module::erase↩
SubformulaFromPassedFormula (
    ModuleInput::iterator _subformula,
    bool _ignoreOrigins = false ) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_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 >.

0.15.99.2.38 excludeNotReceivedVariablesFromModel()

`void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

0.15.99.2.39 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

0.15.99.2.40 firstSubformulaToPass()

`ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.99.2.41 firstUncheckedReceivedSubformula()

`ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.99.2.42 freeSplittingVariable()

```
static void smtrat::Module::freeSplittingVariable (
    const FormulaT & _splittingVariable ) [inline], [static], [inherited]
```

0.15.99.2.43 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.99.2.44 getBackendsAllModels()

`void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.99.2.45 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.99.2.46 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.99.2.47 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.

0.15.99.2.48 getInformationRelevantFormulas() `const std::set< FormulaT > & smtrat::Module↩ ::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.99.2.49 getModelEqualities() `std::list< std::vector< carl::Variable > > smtrat::Module↩ ::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.99.2.50 getOrigins() [1/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.99.2.51 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.99.2.52 getOrigins() [3/3]

```
const FormulaT& smtrat::Module::getOrigins (
    ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]
```

Gets the origins of the passed formula at the given position.

Parameters

_formula	The position of a formula in the passed formulas.

Returns

The origins of the passed formula at the given position.

0.15.99.2.53 getReceivedFormulaSimplified()

```
std::pair< bool, FormulaT > smtrat::Module::getReceivedFormulaSimplified ( ) [virtual], [inherited]
```

Returns

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's reveived formula, if the Boolean is true.

Reimplemented in smtrat::PModule.

0.15.99.2.54 hasLemmas()

```
bool smtrat::Module::hasLemmas ( ) [inline], [inherited]
```

Checks whether this module or any of its backends provides any lemmas.

0.15.99.2.55 hasValidInfeasibleSubset()

```
bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]
```

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.99.2.56 id()

```
std::size_t smtrat::Module::id ( ) const [inline], [inherited]
```

Returns

A unique ID to identify this module instance.

0.15.99.2.57 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasible↩ Subsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.99.2.58 inform() `bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.
It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

Parameters

_constraint	The constraint to inform about.

Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

0.15.99.2.59 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.

0.15.99.2.60 informCore() `virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.99.2.61 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.

0.15.99.2.62 init() `void smtrat::Module::init ( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not be called twice and only before the first runBackends call.
Reimplemented in smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntBlastModule< Settings >, smtrat::FPPModule< Settings >, smtrat::CurryModule< Settings >, and smtrat::BVModule< Settings >.

0.15.99.2.63 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.99.2.64 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.

0.15.99.2.65 isPreprocessor() `bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.99.2.66 lemmas() `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.99.2.67 merge() `std::vector< FormulaT > smtrat::Module::merge ( const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.
({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

Parameters

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns

The vector being the two given vectors merged.

0.15.99.2.68 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.99.2.69 modelsDisjoint()

`bool smtrat::Module::modelsDisjoint ( const Model & _modelA, const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

Parameters

_modelA	The first model to check for.
_modelB	The second model to check for.

Returns

true, if there is no variable assigned by both models; false, otherwise.

0.15.99.2.70 moduleName()

`virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]`

Returns

The name of the given module type as name.

Reimplemented in smtrat::VSModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::STropModule< Settings >, smtrat::SplitSOSModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntEqModule< Settings >, smtrat::IntBlastModule< Settings >, smtrat::IncWidthModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::GBModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::CurryModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::BVModule< Settings >, and smtrat::BEModule< Settings >.

0.15.99.2.71 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

0.15.99.2.72 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.99.2.73 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.99.2.74 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.99.2.75 pPassedFormula() `const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.99.2.76 pReceivedFormula() `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.99.2.77 print() `void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.99.2.78 printAllModels() `void smtrat::Module::printAllModels ( std::ostream & _out = std::cout ) [inherited]`

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.99.2.79 printInfeasibleSubsets() `void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.99.2.80 printModel() `void smtrat::Module::printModel ( std::ostream & _out = std::cout ) const [inherited]`

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.99.2.81 printPassedFormula()

`void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.99.2.82 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.99.2.83 probablyLooping()

`bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.99.2.84 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.99.2.85 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

0.15.99.2.86 receivedVariable()

`bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

0.15.99.2.87 remove()

`void smtrat::Module::remove ( ModuleInput::const_iterator _subformula ) [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.
However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

Parameters

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

0.15.99.2.88 removeCore()

```
virtual void smtrat::Module::removeCore (
    [[maybe_unused] ] 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.

0.15.99.2.89 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin
(
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

0.15.99.2.90 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

0.15.99.2.91 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.99.2.92 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to this module.

0.15.99.2.93 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( ) [inline],
[protected], [virtual], [inherited]
```

Reimplemented in smtrat::PModule.

0.15.99.2.94 runBackends() [2/2]

```
Answer smtrat::Module::runBackends (
    bool _final,
    bool _full,
    carl::Variable _objective ) [protected], [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

Parameters

_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.

0.15.99.2.95 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.

0.15.99.2.96 setThreadPriority()

```
void smtrat::Module::setThreadPriority (
    thread_priority _threadPriority ) [inline], [inherited]
```

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.99.2.97 solverState()

```
Answer smtrat::Module::solverState ( ) const [inline], [inherited]
```

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.99.2.98 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 !=.

0.15.99.2.99 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.99.2.100 updateAllModels() `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.99.2.101 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.99.2.102 updateModel() `void smtrat::Module::updateModel ( ) const [virtual], [inherited]`

Updates the model, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::PModule, smtrat::VSModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::STropModule< Settings >, smtrat::SATModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::MCBModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntEqModule< Settings >, smtrat::IntBlastModule< Settings >, smtrat::IncWidthModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::GBPPModule< Settings >, smtrat::FPPModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::ESModule< Settings >, smtrat::CurryModule< Settings >, smtrat::CubeLIAModule< Se
smtrat::CSplitModule< Settings >, smtrat::CoveringNGModule< Settings >, and smtrat::BVModule< Settings >.

0.15.99.2.103 usedBackends() `const std::vector<Module*>& smtrat::Module::usedBackends ( ) const [inline], [inherited]`

Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.99.3 Field Documentation

0.15.99.3.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.99.3.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.99.3.3 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.

0.15.99.3.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.99.3.5 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

0.15.99.3.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.99.3.7 mFirstSubformulaToPass `ModuleInput::iterator smtrat::Module::mFirstSubformulaTo↩ Pass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.99.3.8 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.

0.15.99.3.9 mFoundAnswer `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.99.3.10 mFullCheck `bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.99.3.11 mInfeasibleSubsets `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.99.3.12 mInformedConstraints `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.99.3.13 mLastBranches `std::vector< Branching > smtrat::Module::mLastBranches = std↩ ::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.99.3.14 mLemmas `std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.99.3.15 mModel `Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.99.3.16 mModelComputed `bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.99.3.17 mNumOfBranchVarsToStore `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.99.3.18 mObjectiveVariable `carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.99.3.19 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

0.15.99.3.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.99.3.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.99.3.22 mSolverState `std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.99.3.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

0.15.99.3.24 mUsedBackends `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.99.3.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.100 smtrat::CollectionWithOrigins< Element, Origin > Class Template Reference

`#include <IntBlastModule.h>`

Public Types

- typedef Super::iterator iterator
- typedef Super::const_iterator const_iterator

Public Member Functions

- CollectionWithOrigins (bool _trackElementsWithoutOrigins=true)
- bool contains (const Element &_element)
- bool add (const Element &_element, const Origin &_origin)
- bool removeOrigin (const Origin &_origin)
- bool removeOrigins (const std::set< Origin > &_origins)
- iterator begin ()
- iterator end ()
- const_iterator cbegin () const
- const_iterator cend () const
- const std::set< Element > & elementsWithoutOrigins () const
- void clearElementsWithoutOrigins ()

0.15.100.1 Member Typedef Documentation

0.15.100.1.1 const_iterator

```
template<typename Element , typename Origin >
typedef Super::const_iterator smtrat::CollectionWithOrigins< Element, Origin >::const_iterator
```

0.15.100.1.2 iterator

```
template<typename Element , typename Origin >
typedef Super::iterator smtrat::CollectionWithOrigins< Element, Origin >::iterator
```

0.15.100.2 Constructor & Destructor Documentation

0.15.100.2.1 CollectionWithOrigins()

```
template<typename Element , typename Origin >
smtrat::CollectionWithOrigins< Element, Origin >::CollectionWithOrigins (
            bool _trackElementsWithoutOrigins = true )  [inline]
```

0.15.100.3 Member Function Documentation

0.15.100.3.1 add()

```
template<typename Element , typename Origin >
bool smtrat::CollectionWithOrigins< Element, Origin >::add (
            const Element & _element,
            const Origin & _origin )  [inline]
```

0.15.100.3.2 begin()

```
template<typename Element , typename Origin >
iterator smtrat::CollectionWithOrigins< Element, Origin >::begin ( )  [inline]
```

0.15.100.3.3 cbegin()

```
template<typename Element , typename Origin >
const_iterator smtrat::CollectionWithOrigins< Element, Origin >::cbegin ( ) const  [inline]
```

0.15.100.3.4 cend()

```
template<typename Element , typename Origin >
const_iterator smtrat::CollectionWithOrigins< Element, Origin >::cend ( ) const  [inline]
```

0.15.100.3.5 clearElementsWithoutOrigins()

```
template<typename Element , typename Origin >
void smtrat::CollectionWithOrigins< Element, Origin >::clearElementsWithoutOrigins ( )  [inline]
```

0.15.100.3.6 contains()

```
template<typename Element , typename Origin >
bool smtrat::CollectionWithOrigins< Element, Origin >::contains (
            const Element & _element )  [inline]
```

0.15.100.3.7 elementsWithoutOrigins()

```
template<typename Element , typename Origin >
const std::set<Element>& smtrat::CollectionWithOrigins< Element, Origin >::elementsWithout↩
Origins ( ) const  [inline]
```

0.15.100.3.8 end()

```
template<typename Element , typename Origin >
iterator smtrat::CollectionWithOrigins< Element, Origin >::end ( )  [inline]
```

0.15.100.3.9 removeOrigin()

```
template<typename Element , typename Origin >
bool smtrat::CollectionWithOrigins< Element, Origin >::removeOrigin (
            const Origin & _origin )  [inline]
```

0.15.100.3.10 removeOrigins()

```
template<typename Element , typename Origin >
bool smtrat::CollectionWithOrigins< Element, Origin >::removeOrigins (
            const std::set< Origin > & _origins )  [inline]
```

0.15.101 smtrat::ICPModule< Settings >::comp Struct Reference

Typedefs:

`#include <ICPModule.h>`

Public Member Functions

- bool operator() (std::pair< double, unsigned > lhs, std::pair< double, unsigned > rhs) const

0.15.101.1 Detailed Description

template<class Settings>
struct smtrat::ICPModule< Settings >::comp

Typedefs:

0.15.101.2 Member Function Documentation

0.15.101.2.1 operator()()

```
template<class Settings >
bool smtrat::ICPModule< Settings >::comp::operator() (
            std::pair< double, unsigned > lhs,
            std::pair< double, unsigned > rhs ) const  [inline]
```

0.15.102 smtrat::cadcells::datastructures::CompoundMaxMin Struct Reference

Represents the maximum function of the contained indexed root functions.

`#include <roots.h>`

Public Member Functions

- void polys (boost::container::flat_set< PolyRef > &result) const

Data Fields

- std::vector< std::vector< IndexedRoot > > roots

0.15.102.1 Detailed Description

Represents the maximum function of the contained indexed root functions.

0.15.102.2 Member Function Documentation

0.15.102.2.1 polys()

```
void smtrat::cadcells::datastructures::CompoundMaxMin::polys (
            boost::container::flat_set< PolyRef > & result ) const  [inline]
```

0.15.102.3 Field Documentation

0.15.102.3.1 roots

```
std::vector<std::vector<IndexedRoot> > smtrat::cadcells::datastructures↩
::CompoundMaxMin::roots
```

0.15.103 smtrat::cadcells::datastructures::CompoundMinMax Struct Reference

Represents the minimum function of the contained indexed root functions.

`#include <roots.h>`

Public Member Functions

- void polys (boost::container::flat_set< PolyRef > &result) const

Data Fields

- std::vector< std::vector< IndexedRoot > > roots

0.15.103.1 Detailed Description

Represents the minimum function of the contained indexed root functions.

0.15.103.2 Member Function Documentation

0.15.103.2.1 polys()

```
void smtrat::cadcells::datastructures::CompoundMinMax::polys (
            boost::container::flat_set< PolyRef > & result ) const  [inline]
```

0.15.103.3 Field Documentation

0.15.103.3.1 roots `std::vector<std::vector<IndexedRoot> > smtrat::cadcells::datastructures::CompoundMinMax::roots`

0.15.104 smtrat::vs::Condition Class Reference

`#include <Condition.h>`

Public Member Functions

- Condition (const smtrat::ConstraintT &, size_t _id, size_t=0, bool=false, const carl::PointerSet< Condition > &=carl::PointerSet< Condition >(), bool=false)
- Condition (const Condition &, size_t _id)
- Condition (const Condition &)=delete
- ~Condition ()
- bool & rFlag () const
- bool flag () const
- bool & rRecentlyAdded () const
- bool recentlyAdded () const
- size_t & rValuation () const
- size_t valuation () const
- size_t id () const
- const smtrat::ConstraintT & constraint () const
- carl::PointerSet< Condition > * pOriginalConditions () const
- const carl::PointerSet< Condition > & originalConditions () const
- double valuate (const carl::Variable &, size_t, bool) const

  Valuates the constraint according to a variable (it possibly not contains).
- bool operator== (const Condition &) const

  Checks the equality of a given condition (right hand side) with this condition (left hand side).
- bool operator< (const Condition &) const

  Checks if the given condition (right hand side) is greater than this condition (left hand side).
- void print (std::ostream &=std::cout) const

  Prints the condition to an output stream.

Friends

- std::ostream & operator<< (std::ostream &_out, const Condition &_condition)

0.15.104.1 Constructor & Destructor Documentation

0.15.104.1.1 Condition() [1/3]

```
smtrat::vs::Condition::Condition (
    const smtrat::ConstraintT & _cons,
    size_t _id,
    size_t _val = 0,
    bool _flag = false,
    const carl::PointerSet< Condition > & _oConds = carl::PointerSet<Condition>(),
    bool _rAdded = false )
```

0.15.104.1.2 Condition() [2/3]

```
smtrat::vs::Condition::Condition (
    const Condition & _cond,
    size_t _id )
```

0.15.104.1.3 Condition() [3/3]

```
smtrat::vs::Condition::Condition (
    const Condition & ) [delete]
```

0.15.104.1.4 ~Condition()

```
smtrat::vs::Condition::~Condition ( )
```

0.15.104.2 Member Function Documentation

0.15.104.2.1 constraint()

```
const smtrat::ConstraintT& smtrat::vs::Condition::constraint ( ) const [inline]
```

0.15.104.2.2 flag()

```
bool smtrat::vs::Condition::flag ( ) const [inline]
```

0.15.104.2.3 id()

```
size_t smtrat::vs::Condition::id ( ) const [inline]
```

0.15.104.2.4 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.

0.15.104.2.5 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.

0.15.104.2.6 originalConditions()

```
const carl::PointerSet<Condition>& smtrat::vs::Condition↩
::originalConditions ( ) const [inline]
```

0.15.104.2.7 pOriginalConditions()

`carl::PointerSet<Condition>* smtrat::vs::Condition::pOriginalConditions ( ) const [inline]`

0.15.104.2.8 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.

0.15.104.2.9 recentlyAdded()

`bool smtrat::vs::Condition::recentlyAdded ( ) const [inline]`

0.15.104.2.10 rFlag()

`bool& smtrat::vs::Condition::rFlag ( ) const [inline]`

0.15.104.2.11 rRecentlyAdded()

`bool& smtrat::vs::Condition::rRecentlyAdded ( ) const [inline]`

0.15.104.2.12 rValuation()

`size_t& smtrat::vs::Condition::rValuation ( ) const [inline]`

0.15.104.2.13 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.

0.15.104.2.14 valuation()

`size_t smtrat::vs::Condition::valuation ( ) const [inline]`

0.15.104.3 Friends And Related Function Documentation

0.15.104.3.1 operator<<

```
std::ostream& operator<< (
    std::ostream & _out,
    const Condition & _condition ) [friend]
```

0.15.105 benchmax::CondorBackend Class Reference

Backend for the HTCondor batch system.

`#include <CondorBackend.h>`

Public Member Functions

- void run (const Jobs &, bool)

  Run all tools on all benchmarks.
- bool suspendable () const
- void process_results (const Jobs &jobs, bool check_finished)
- void addResult (const Tool *tool, const fs::path &file, BenchmarkResult &&result)

  Add a result.

Protected Member Functions

- virtual void execute (const Tool *, const fs::path &)

  No-op version of execute.
- virtual void startTool (const Tool *)

  Hook for every tool at the beginning.
- virtual void finalize ()

  Hook to allow for asynchronous backends to wait for jobs to terminate.
- void madeProgress (std::size_t files=1)

  Can be called to give information about the current progress, if available.
- virtual bool collect_results (const Jobs &, bool)
- void sanitize_results (const Jobs &jobs) const
- void write_results (const Jobs &jobs) const

Protected Attributes

- std::size_t mExpectedJobs

  Number of jobs that should be run.
- std::atomic< std::size_t > mFinishedJobs

  Number of jobs that are finished.
- std::atomic< std::size_t > mLastPercent

  Percentage of finished jobs when madeProgress() was last called.

0.15.105.1 Detailed Description

Backend for the HTCondor batch system.
Currently submits all jobs individually and asynchronously waits for them to finish.

0.15.105.2 Member Function Documentation

0.15.105.2.1 addResult()

```
void benchmax::Backend::addResult (
    const Tool * tool,
    const fs::path & file,
    BenchmarkResult && result ) [inline], [inherited]
```

Add a result.

0.15.105.2.2 collect_results()

```
virtual bool benchmax::Backend::collect_results (
    const Jobs & ,
    bool ) [inline], [protected], [virtual], [inherited]
```

0.15.105.2.3 execute()

```
virtual void benchmax::CondorBackend::execute (
    const Tool * ,
    const fs::path & ) [inline], [protected], [virtual]
```

No-op version of execute.
Reimplemented from benchmax::Backend.

0.15.105.2.4 finalize()

```
virtual void benchmax::Backend::finalize ( ) [inline], [protected], [virtual], [inherited]
```

Hook to allow for asynchronous backends to wait for jobs to terminate.

0.15.105.2.5 madeProgress()

```
void benchmax::Backend::madeProgress (
    std::size_t files = 1 ) [inline], [protected], [inherited]
```

Can be called to give information about the current progress, if available.

0.15.105.2.6 process_results()

```
void benchmax::Backend::process_results (
    const Jobs & jobs,
    bool check_finished ) [inline], [inherited]
```

0.15.105.2.7 run()

```
void benchmax::CondorBackend::run (
    const Jobs & ,
    bool ) [inline]
```

Run all tools on all benchmarks.

0.15.105.2.8 sanitize_results()

```
void benchmax::Backend::sanitize_results (
    const Jobs & jobs ) const [inline], [protected], [inherited]
```

0.15.105.2.9 startTool()

```
virtual void benchmax::Backend::startTool (
    const Tool * ) [inline], [protected], [virtual], [inherited]
```

Hook for every tool at the beginning.
Can be used to upload the tool to some remote system.

0.15.105.2.10 suspendable()

```
bool benchmax::Backend::suspendable ( ) const [inline], [inherited]
```

0.15.105.2.11 write_results()

```
void benchmax::Backend::write_results (
    const Jobs & jobs ) const [inline], [protected], [inherited]
```

0.15.105.3 Field Documentation

0.15.105.3.1 mExpectedJobs

```
std::size_t benchmax::Backend::mExpectedJobs [protected], [inherited]
```

Number of jobs that should be run.

0.15.105.3.2 mFinishedJobs

```
std::atomic<std::size_t> benchmax::Backend::mFinishedJobs [protected], [inherited]
```

Number of jobs that are finished.

0.15.105.3.3 mLastPercent `std::atomic<std::size_t> benchmax::Backend::mLastPercent [protected], [inherited]`

Percentage of finished jobs when madeProgress() was last called.

0.15.106 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)

0.15.106.1 Constructor & Destructor Documentation

0.15.106.1.1 ConflictGenerator() `template<class Comparator >`

```
smtrat::mcsat::fm::ConflictGenerator< Comparator >::ConflictGenerator (
            const std::vector< ConstraintT > & bounds,
            const Model & m,
            carl::Variable v ) [inline]
```

0.15.106.2 Member Function Documentation

0.15.106.2.1 generateExplanation() `template<class Comparator >`

```
template<typename Callback >
void smtrat::mcsat::fm::ConflictGenerator< Comparator >::generateExplanation (
            Callback && callback ) [inline]
```

0.15.107 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)

0.15.107.1 Detailed Description

Representation of a bipartite graph (C+S, E) of vertices C and S, representing the constraints and samples, respectively.
A vertex from C and a vertex from S are connected by an edge in E iff the corresponding constraint conflicts with the corresponding sample point.

0.15.107.2 Constructor & Destructor Documentation

0.15.107.2.1 ConflictGraph()

`smtrat::cad::ConflictGraph::ConflictGraph ( std::size_t constraints ) [inline]`

0.15.107.3 Member Function Documentation

0.15.107.3.1 addSample()

`void smtrat::cad::ConflictGraph::addSample ( const Sample & sample )`

0.15.107.3.2 coveredSamples()

`std::size_t smtrat::cad::ConflictGraph::coveredSamples ( std::size_t id ) const [inline]`

0.15.107.3.3 getData()

`carl::Bitset smtrat::cad::ConflictGraph::getData ( std::size_t id )`

0.15.107.3.4 getMaxDegreeConstraint()

`std::size_t smtrat::cad::ConflictGraph::getMaxDegreeConstraint ( ) const`

Retrieves the constraint that covers the most samples.

0.15.107.3.5 getRemainingConstraints()

`std::vector< std::pair< std::size_t, carl::Bitset > > smtrat::cad::ConflictGraph::getRemainingConstraints ( )`

0.15.107.3.6 hasRemainingSamples()

`bool smtrat::cad::ConflictGraph::hasRemainingSamples ( ) const`

Checks if there are samples still uncovered.

0.15.107.3.7 numRemainingConstraints()

`std::size_t smtrat::cad::ConflictGraph::numRemainingConstraints ( ) const`

0.15.107.3.8 numSamples() `std::size_t smtrat::cad::ConflictGraph::numSamples ( ) const`

0.15.107.3.9 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.

0.15.107.3.10 selectConstraint() `void smtrat::cad::ConflictGraph::selectConstraint ( std::size_t id )`

Removes the given constraint and disable all sample points covered by this constraint.

0.15.107.3.11 selectEssentialConstraints() `std::vector< std::size_t > smtrat::cad::Conflict↩ Graph::selectEssentialConstraints ( )`

Selects all essential constraints and returns their indices.

0.15.107.4 Friends And Related Function Documentation

0.15.107.4.1 operator<< `std::ostream& operator<< ( std::ostream & os, const ConflictGraph & cg ) [friend]`

0.15.108 smtrat::parser::Theories::ConstantAdder Struct Reference

Helper functor for addConstants() method.

`#include <Theories.h>`

Public Member Functions

- ConstantAdder (qi::symbols< char, types::ConstType > &constants)
- template<typename T >
  void operator() (T *)

Data Fields

- qi::symbols< char, types::ConstType > & constants

0.15.108.1 Detailed Description

Helper functor for addConstants() method.

0.15.108.2 Constructor & Destructor Documentation

0.15.108.2.1 ConstantAdder() `smtrat::parser::Theories::ConstantAdder::ConstantAdder ( qi::symbols< char, types::ConstType > & constants ) [inline]`

0.15.108.3 Member Function Documentation

0.15.108.3.1 operator()() `template<typename T >`

```
void smtrat::parser::Theories::ConstantAdder::operator() (
            T * ) [inline]
```

0.15.108.4 Field Documentation

0.15.108.4.1 constants `qi::symbols<char, types::ConstType>& smtrat::parser::Theories::Constant↩ Adder::constants`

0.15.109 smtrat::covering_ng::formula::node_ds::CONSTRAINT Struct Reference

`#include <FormulaEvaluation.h>`

Data Fields

- carl::BasicConstraint< cadcells::Polynomial > constraint

0.15.109.1 Field Documentation

0.15.109.1.1 constraint `carl::BasicConstraint<cadcells::Polynomial> smtrat::covering_ng::formula↩ ::node_ds::CONSTRAINT::constraint`

0.15.110 smtrat::cad::CADConstraints< BT >::ConstraintComparator Struct Reference

`#include <CADConstraints.h>`

Public Member Functions

- std::size_t complexity (const ConstraintT &c) const
- bool operator() (const ConstraintT &lhs, const ConstraintT &rhs) const

0.15.110.1 Member Function Documentation

0.15.110.1.1 complexity() `template<Backtracking BT>`

```
std::size_t smtrat::cad::CADConstraints< BT >::ConstraintComparator::complexity (
            const ConstraintT & c ) const [inline]
```

0.15.110.1.2 operator()() `template<Backtracking BT>`

```
bool smtrat::cad::CADConstraints< BT >::ConstraintComparator::operator() (
            const ConstraintT & lhs,
            const ConstraintT & rhs ) const [inline]
```

0.15.111 smtrat::cad::preprocessor::ConstraintUpdate Struct Reference

`#include <CADPreprocessor.h>`

Data Fields

- std::vector< ConstraintT > toAdd
- std::vector< ConstraintT > toRemove

0.15.111.1 Field Documentation

0.15.111.1.1 toAdd `std::vector< ConstraintT > smtrat::cad::preprocessor::ConstraintUpdate↩ ::toAdd`

0.15.111.1.2 toRemove `std::vector< ConstraintT > smtrat::cad::preprocessor::Constraint↩ Update::toRemove`

0.15.112 smtrat::ConstrTree Class Reference

`#include <IntBlastModule.h>`

Public Member Functions

- ConstrTree (const ConstraintT &_constraint)
- ~ConstrTree ()
- carl::Relation relation () const
- const PolyTree & left () const
- const PolyTree & right () const
- const ConstraintT & constraint () const

0.15.112.1 Constructor & Destructor Documentation

0.15.112.1.1 ConstrTree() `smtrat::ConstrTree::ConstrTree ( const ConstraintT & _constraint ) [inline]`

0.15.112.1.2 ~ConstrTree() `smtrat::ConstrTree::~ConstrTree ( ) [inline]`

0.15.112.2 Member Function Documentation

0.15.112.2.1 constraint() `const ConstraintT& smtrat::ConstrTree::constraint ( ) const [inline]`

0.15.112.2.2 left() `const PolyTree& smtrat::ConstrTree::left ( ) const [inline]`

0.15.112.2.3 relation() `carl::Relation smtrat::ConstrTree::relation ( ) const [inline]`

0.15.112.2.4 right() `const PolyTree& smtrat::ConstrTree::right ( ) const [inline]`

0.15.113 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.

0.15.113.1 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.

0.15.113.2 Constructor & Destructor Documentation

0.15.113.2.1 Context() `template<class Settings >`

```
smtrat::LRAModule< Settings >::Context::Context (
            const FormulaT & _origin,
            ModuleInput::iterator _pos ) [inline]
```

0.15.113.3 Field Documentation

0.15.113.3.1 origin `template<class Settings >`

`FormulaT smtrat::LRAModule< Settings >::Context::origin`

The formula in the received formula.

0.15.113.3.2 position `template<class Settings >`

`ModuleInput::iterator smtrat::LRAModule< Settings >::Context::position`

The position of this formula in the passed formula.

0.15.114 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 is_linear () const
- unsigned reusagesAfterTargetDiameterReached () const
- unsigned incrementReusagesAfterTargetDiameterReached ()
- void resetReusagesAfterTargetDiameterReached ()
- void addICPVariable (IcpVariable *_icpVar)
- double calcRWA ()
- double RWA () const
- double lastRWA () const
- void updateLastRWA ()
- double lastPayoff () const
- unsigned id () const
- void setDerivationVar (carl::Variable _var)
- void setLhs (carl::Variable _lhs)
- void setPayoff (double _weight)
- void calcDerivative () throw ()
- size_t activity ()
- bool isActive () const
- bool isDerived () const
- void resetWeights ()
- void print (std::ostream &_out=std::cout) const
- bool operator< (ContractionCandidate const &rhs) const
- bool operator== (ContractionCandidate const &rhs) const

0.15.114.1 Constructor & Destructor Documentation

0.15.114.1.1 ContractionCandidate() [1/4]

```
smtrat::icp::ContractionCandidate::ContractionCandidate (
    const ContractionCandidate & _original ) [delete]
```

Constructors:

0.15.114.1.2 ContractionCandidate() [2/4]

```
smtrat::icp::ContractionCandidate::ContractionCandidate ( ) [delete]
```

0.15.114.1.3 ContractionCandidate() [3/4]

```
smtrat::icp::ContractionCandidate::ContractionCandidate (
    carl::Variable _lhs,
    const Poly _rhs,
    const ConstraintT & _constraint,
    carl::Variable _derivationVar,
    Contractor< carl::SimpleNewton > & _contractor,
    const FormulaT & _origin,
    unsigned _id,
    bool _usePropagation ) [inline]
```

0.15.114.1.4 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	

0.15.114.1.5 ~ContractionCandidate()

```
smtrat::icp::ContractionCandidate::~ContractionCandidate ( ) [inline]
```

Destructor:

0.15.114.2 Member Function Documentation

0.15.114.2.1 activity()

```
size_t smtrat::icp::ContractionCandidate::activity ( ) [inline]
```

0.15.114.2.2 addICPVariable()

```
void smtrat::icp::ContractionCandidate::addICPVariable (
    IcpVariable * _icpVar ) [inline]
```

0.15.114.2.3 addOrigin()

```
void smtrat::icp::ContractionCandidate::addOrigin (
    const FormulaT & _origin )
```

0.15.114.2.4 calcDerivative()

```
void smtrat::icp::ContractionCandidate::calcDerivative ( ) throw ( ) [inline]
```

0.15.114.2.5 calcRWA()

```
double smtrat::icp::ContractionCandidate::calcRWA ( ) [inline]
```

0.15.114.2.6 constraint()

```
const ConstraintT& smtrat::icp::ContractionCandidate::constraint ( ) const [inline]
```

0.15.114.2.7 contract()

```
bool smtrat::icp::ContractionCandidate::contract (
    EvalDoubleIntervalMap & _intervals,
    DoubleInterval & _resA,
    DoubleInterval & _resB ) [inline]
```

0.15.114.2.8 contractor()

```
Contractor<carl::SimpleNewton>& smtrat::icp::ContractionCandidate↩
::contractor ( ) const [inline]
```

0.15.114.2.9 derivationVar()

```
carl::Variable::Arg smtrat::icp::ContractionCandidate::derivation↩
Var ( ) const [inline]
```

0.15.114.2.10 derivative()

```
const Poly& smtrat::icp::ContractionCandidate::derivative ( ) const
[inline]
```

0.15.114.2.11 hasOrigin()

```
bool smtrat::icp::ContractionCandidate::hasOrigin (
    const FormulaT & _origin ) const [inline]
```

0.15.114.2.12 id()

```
unsigned smtrat::icp::ContractionCandidate::id ( ) const [inline]
```

0.15.114.2.13 incrementReusagesAfterTargetDiameterReached()

```
unsigned smtrat::icp::Contraction↩
Candidate::incrementReusagesAfterTargetDiameterReached ( ) [inline]
```

0.15.114.2.14 is_linear()

```
bool smtrat::icp::ContractionCandidate::is_linear ( ) const [inline]
```

0.15.114.2.15 isActive()

```
bool smtrat::icp::ContractionCandidate::isActive ( ) const [inline]
```

0.15.114.2.16 isDerived()

```
bool smtrat::icp::ContractionCandidate::isDerived ( ) const [inline]
```

0.15.114.2.17 lastPayoff()

```
double smtrat::icp::ContractionCandidate::lastPayoff ( ) const [inline]
```

0.15.114.2.18 lastRWA()

```
double smtrat::icp::ContractionCandidate::lastRWA ( ) const [inline]
```

0.15.114.2.19 lhs()

```
carl::Variable::Arg smtrat::icp::ContractionCandidate::lhs ( ) const [inline]
```

0.15.114.2.20 operator<()

```
bool smtrat::icp::ContractionCandidate::operator< (
    ContractionCandidate const & rhs ) const [inline]
```

0.15.114.2.21 operator==()

`bool smtrat::icp::ContractionCandidate::operator== ( ContractionCandidate const & rhs ) const [inline]`

0.15.114.2.22 origin()

`const FormulaSetT& smtrat::icp::ContractionCandidate::origin ( ) const [inline]`

0.15.114.2.23 print()

`void smtrat::icp::ContractionCandidate::print ( std::ostream & _out = std::cout ) const [inline]`

0.15.114.2.24 removeOrigin()

`void smtrat::icp::ContractionCandidate::removeOrigin ( const FormulaT & _origin )`

0.15.114.2.25 resetReusagesAfterTargetDiameterReached()

`void smtrat::icp::ContractionCandidate↩ ::resetReusagesAfterTargetDiameterReached ( ) [inline]`

0.15.114.2.26 resetWeights()

`void smtrat::icp::ContractionCandidate::resetWeights ( ) [inline]`

0.15.114.2.27 reusagesAfterTargetDiameterReached()

`unsigned smtrat::icp::ContractionCandidate↩ ::reusagesAfterTargetDiameterReached ( ) const [inline]`

0.15.114.2.28 rhs()

`const Poly& smtrat::icp::ContractionCandidate::rhs ( ) const [inline]`

Functions:

0.15.114.2.29 rOrigin()

`FormulaSetT& smtrat::icp::ContractionCandidate::rOrigin ( ) [inline]`

0.15.114.2.30 RWA()

`double smtrat::icp::ContractionCandidate::RWA ( ) const [inline]`

0.15.114.2.31 setDerivationVar()

`void smtrat::icp::ContractionCandidate::setDerivationVar ( carl::Variable _var ) [inline]`

0.15.114.2.32 setLhs()

`void smtrat::icp::ContractionCandidate::setLhs ( carl::Variable _lhs ) [inline]`

0.15.114.2.33 setLinear()

`void smtrat::icp::ContractionCandidate::setLinear ( ) [inline]`

0.15.114.2.34 setNonlinear()

`void smtrat::icp::ContractionCandidate::setNonlinear ( ) [inline]`

0.15.114.2.35 setPayoff()

`void smtrat::icp::ContractionCandidate::setPayoff ( double _weight ) [inline]`

0.15.114.2.36 updateLastRWA() `void smtrat::icp::ContractionCandidate::updateLastRWA ( ) [inline]`

0.15.115 smtrat::icp::contractionCandidateComp Struct Reference

`#include <ContractionCandidate.h>`

Public Member Functions

- bool operator() (const ContractionCandidate *const lhs, const ContractionCandidate *const rhs) const

0.15.115.1 Member Function Documentation

0.15.115.1.1 operator()() `bool smtrat::icp::contractionCandidateComp::operator() ( const ContractionCandidate *const lhs, const ContractionCandidate *const rhs ) const [inline]`

0.15.116 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 ()

0.15.116.1 Constructor & Destructor Documentation

0.15.116.1.1 ContractionCandidateManager() `smtrat::icp::ContractionCandidateManager::Contraction↩ CandidateManager ( )`

Constructors.

0.15.116.1.2 ~ContractionCandidateManager() `smtrat::icp::ContractionCandidateManager::~↩ ContractionCandidateManager ( ) [inline]`

0.15.116.2 Member Function Documentation

0.15.116.2.1 candidates()

```
const std::vector<ContractionCandidate*>& smtrat::icp::ContractionCandidateManager::candidates ( ) [inline]
```

0.15.116.2.2 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

0.15.116.2.3 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

0.15.116.2.4 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

0.15.116.2.5 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

0.15.117 smtrat::parser::conversion::Converter< To > Struct Template Reference

```
#include <Conversions.h>
```

Public Member Functions

- template<typename From >
  bool operator() (const From &, To &) const
- bool operator() (const To &from, To &to) const

0.15.117.1 Member Function Documentation

0.15.117.1.1 operator()() [1/2]

```
template<typename To >
template<typename From >
bool smtrat::parser::conversion::Converter< To >::operator() (
            const From & ,
            To &  ) const  [inline]
```

0.15.117.1.2 operator()() [2/2]

```
template<typename To >
bool smtrat::parser::conversion::Converter< To >::operator() (
            const To & from,
            To & to ) const  [inline]
```

0.15.118 smtrat::parser::conversion::Converter< FormulaT > Struct Reference

```
#include <Conversions.h>
```

Public Member Functions

- template<typename From >
  bool operator() (const From &, FormulaT &) const
- bool operator() (const FormulaT &from, FormulaT &to) const
- bool operator() (const carl::Variable &from, FormulaT &to) const

0.15.118.1 Member Function Documentation

0.15.118.1.1 operator()() [1/3]

```
bool smtrat::parser::conversion::Converter< FormulaT >::operator()
(
            const carl::Variable & from,
            FormulaT & to ) const  [inline]
```

0.15.118.1.2 operator()() [2/3]

```
bool smtrat::parser::conversion::Converter< FormulaT >::operator()
(
            const FormulaT & from,
            FormulaT & to ) const  [inline]
```

0.15.118.1.3 operator()() [3/3]

```
template<typename From >
bool smtrat::parser::conversion::Converter< FormulaT >::operator() (
            const From & ,
            FormulaT &  ) const  [inline]
```

0.15.119 smtrat::parser::conversion::Converter< Poly > Struct Reference

```
#include <Conversions.h>
```

Public Member Functions

- template<typename From >
  bool operator() (const From &, Poly &) const
- bool operator() (const Poly &from, Poly &to) const
- bool operator() (const carl::Variable &from, Poly &to) const
- bool operator() (const Rational &from, Poly &to) const

0.15.119.1 Member Function Documentation

0.15.119.1.1 operator()() [1/4]

```
bool smtrat::parser::conversion::Converter< Poly >::operator() (
            const carl::Variable & from,
            Poly & to ) const  [inline]
```

0.15.119.1.2 operator()() [2/4]

```
template<typename From >
bool smtrat::parser::conversion::Converter< Poly >::operator() (
            const From & ,
            Poly &  ) const  [inline]
```

0.15.119.1.3 operator()() [3/4]

```
bool smtrat::parser::conversion::Converter< Poly >::operator() (
            const Poly & from,
            Poly & to ) const  [inline]
```

0.15.119.1.4 operator()() [4/4]

```
bool smtrat::parser::conversion::Converter< Poly >::operator() (
            const Rational & from,
            Poly & to ) const  [inline]
```

0.15.120 smtrat::parser::conversion::Converter< types::BVTerm > Struct Reference

```
#include <Conversions.h>
```

Public Member Functions

- template<typename From >
  bool operator() (const From &, types::BVTerm &) const
- bool operator() (const types::BVVariable &from, types::BVTerm &to) const
- bool operator() (const types::BVTerm &from, types::BVTerm &to) const
- bool operator() (const FixedWidthConstant< Integer > &from, types::BVTerm &to) const

0.15.120.1 Member Function Documentation

0.15.120.1.1 operator()() [1/4]

```
bool smtrat::parser::conversion::Converter< types::BVTerm >↩
::operator() (
            const FixedWidthConstant< Integer > & from,
            types::BVTerm & to ) const [inline]
```

0.15.120.1.2 operator()() [2/4]

```
template<typename From >
bool smtrat::parser::conversion::Converter< types::BVTerm >::operator() (
            const From & ,
            types::BVTerm & ) const [inline]
```

0.15.120.1.3 operator()() [3/4]

```
bool smtrat::parser::conversion::Converter< types::BVTerm >↩
::operator() (
            const types::BVTerm & from,
            types::BVTerm & to ) const [inline]
```

0.15.120.1.4 operator()() [4/4]

```
bool smtrat::parser::conversion::Converter< types::BVTerm >↩
::operator() (
            const types::BVVariable & from,
            types::BVTerm & to ) const [inline]
```

0.15.121 benchmax::settings::CoreSettings Struct Reference

Core settings.

```
#include <Settings.h>
```

Data Fields

- bool show_help

  Whether to show the command-line help.
- bool show_settings

  Whether to show the configured settings.
- bool be_verbose

  Whether to be more verbose.
- bool be_quiet

  Whether to be more quiet.
- std::string config_file

  Path of an optional config file.
- long start_time = std::time(nullptr)

  Timestamp the binary was started.

0.15.121.1 Detailed Description

Core settings.

0.15.121.2 Field Documentation

0.15.121.2.1 be_quiet `bool benchmax::settings::CoreSettings::be_quiet`

Whether to be more quiet.

0.15.121.2.2 be_verbose `bool benchmax::settings::CoreSettings::be_verbose`

Whether to be more verbose.

0.15.121.2.3 config_file `std::string benchmax::settings::CoreSettings::config_file`

Path of an optional config file.

0.15.121.2.4 show_help `bool benchmax::settings::CoreSettings::show_help`

Whether to show the command-line help.

0.15.121.2.5 show_settings `bool benchmax::settings::CoreSettings::show_settings`

Whether to show the configured settings.

0.15.121.2.6 start_time `long benchmax::settings::CoreSettings::start_time = std::time(nullptr)`

Timestamp the binary was started.

0.15.122 smtrat::settings::CoreSettings Struct Reference

`#include <Settings.h>`

Data Fields

- bool show_help
- bool show_info
- bool show_version
- bool show_settings
- bool show_cmake_options
- bool show_strategy
- bool show_license
- std::string config_file

0.15.122.1 Field Documentation

0.15.122.1.1 config_file `std::string smtrat::settings::CoreSettings::config_file`

0.15.122.1.2 show_cmake_options `bool smtrat::settings::CoreSettings::show_cmake_options`

0.15.122.1.3 show_help `bool smtrat::settings::CoreSettings::show_help`

0.15.122.1.4 show_info `bool smtrat::settings::CoreSettings::show_info`

0.15.122.1.5 show_license `bool smtrat::settings::CoreSettings::show_license`

0.15.122.1.6 show_settings `bool smtrat::settings::CoreSettings::show_settings`

0.15.122.1.7 show_strategy `bool smtrat::settings::CoreSettings::show_strategy`

0.15.122.1.8 show_version `bool smtrat::settings::CoreSettings::show_version`

0.15.123 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

0.15.123.1 Detailed Description

Implements the core theory of the booleans.

0.15.123.2 Member Typedef Documentation

0.15.123.2.1 OperatorType

```
using smtrat::parser::CoreTheory::OperatorType = boost::variant<carl↩
::FormulaType>
```

0.15.123.3 Constructor & Destructor Documentation

0.15.123.3.1 CoreTheory()

```
smtrat::parser::CoreTheory::CoreTheory (
    ParserState * state )
```

0.15.123.4 Member Function Documentation

0.15.123.4.1 addConstants()

```
void smtrat::parser::CoreTheory::addConstants (
    qi::symbols< char, types::ConstType > & constants )  [static]
```

0.15.123.4.2 addSimpleSorts()

```
void smtrat::parser::CoreTheory::addSimpleSorts (
    qi::symbols< char, carl::Sort > & sorts )  [static]
```

0.15.123.4.3 declareVariable()

```
bool smtrat::parser::CoreTheory::declareVariable (
    const std::string & ,
    const carl::Sort & ,
    types::VariableType & ,
    TheoryError & errors )  [virtual]
```

Declare a new variable with the given name and the given sort.
Reimplemented from smtrat::parser::AbstractTheory.

0.15.123.4.4 expandDistinct()

```
template<typename T , typename Builder >
FormulaT smtrat::parser::AbstractTheory::expandDistinct (
    const std::vector< T > & values,
    const Builder & neqBuilder )  [inline], [inherited]
```

0.15.123.4.5 functionCall()

```
bool smtrat::parser::CoreTheory::functionCall (
    const Identifier & ,
    const std::vector< types::TermType > & ,
    types::TermType & ,
    TheoryError & errors )  [virtual]
```

Resolve another unknown function call.
Reimplemented from smtrat::parser::AbstractTheory.

0.15.123.4.6 handleDistinct()

```
bool smtrat::parser::CoreTheory::handleDistinct (
    const std::vector< types::TermType > & ,
    types::TermType & ,
    TheoryError & errors )  [virtual]
```

Resolve a distinct operator.
Reimplemented from smtrat::parser::AbstractTheory.

0.15.123.4.7 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.

0.15.123.4.8 instantiate()

```
virtual bool smtrat::parser::AbstractTheory::instantiate (
    const types::VariableType & ,
    const types::TermType & ,
    types::TermType & ,
    TheoryError & errors )  [inline], [virtual], [inherited]
```

Instantiate a variable within a term.
Reimplemented in smtrat::parser::BitvectorTheory, and smtrat::parser::ArithmeticTheory.

0.15.123.4.9 refreshVariable()

```
virtual bool smtrat::parser::AbstractTheory::refreshVariable (
    const types::VariableType & ,
    types::VariableType & ,
    TheoryError & errors )  [inline], [virtual], [inherited]
```

Reimplemented in smtrat::parser::BitvectorTheory.

0.15.123.4.10 resolveSymbol()

```
virtual bool smtrat::parser::AbstractTheory::resolveSymbol (
    const Identifier & ,
    types::TermType & ,
    TheoryError & errors )  [inline], [virtual], [inherited]
```

Resolve a symbol that was not declared within the ParserState.
This might be a symbol that actually uses indices, for example bitvector constants.
Reimplemented in smtrat::parser::BitvectorTheory.

0.15.123.5 Field Documentation

0.15.123.5.1 state

```
ParserState* smtrat::parser::AbstractTheory::state  [inherited]
```

0.15.124 smtrat::parser::types::CoreTheory Struct Reference

Types of the core theory.

```
#include <TheoryTypes.h>
```

Public Types

- typedef mpl::vector< FormulaT, std::string > ConstTypes
- typedef mpl::vector< carl::Variable > VariableTypes

- typedef mpl::vector< FormulaT, std::string > ExpressionTypes
- typedef mpl::vector< FormulaT, std::string > TermTypes
- typedef carl::mpl_variant_of< TermTypes >::type TermType

0.15.124.1 Detailed Description

Types of the core theory.

0.15.124.2 Member Typedef Documentation

0.15.124.2.1 ConstTypes `typedef mpl::vector<FormulaT, std::string> smtrat::parser::types::CoreTheory::ConstTy`

0.15.124.2.2 ExpressionTypes `typedef mpl::vector<FormulaT, std::string> smtrat::parser::types::CoreTheory::Ex`

0.15.124.2.3 TermType `typedef carl::mpl_variant_of<TermTypes>::type smtrat::parser::types::CoreTheory::TermT`

0.15.124.2.4 TermTypes `typedef mpl::vector<FormulaT, std::string> smtrat::parser::types::CoreTheory::TermTyp`

0.15.124.2.5 VariableTypes `typedef mpl::vector<carl::Variable> smtrat::parser::types::CoreTheory::VariableType`

0.15.125 smtrat::cadcells::representation::covering< H > Struct Template Reference

`#include <heuristics.h>`

Static Public Member Functions

- template<typename T >
  static std::optional< datastructures::CoveringRepresentation< T > > compute (const std::vector< datastructures::SampledDerivationRef< T >> &ders)

0.15.125.1 Member Function Documentation

0.15.125.1.1 compute()

```
template<CoveringHeuristic H>
template<typename T >
static std::optional<datastructures::CoveringRepresentation<T> > smtrat::cadcells::representation::covering<
H >::compute (
            const std::vector< datastructures::SampledDerivationRef< T >> & ders )  [static]
```

0.15.126 smtrat::mcsat::arithmetic::Covering Class Reference

Semantics: The space is divided into a number of intervals: (-oo,a)[a,a](a,b)[b,b](b,oo) A bit is set if the constraints refutes the corresponding interval.

`#include <Covering.h>`

Public Member Functions

- Covering (std::size_t intervals)
- void add (const FormulaT &c, const carl::Bitset &b)
- bool conflicts () const
- std::size_t satisfyingInterval () const
- const auto & satisfyingSamples () const
- void buildConflictingCore (std::vector< FormulaT > &core) const

Friends

- std::ostream & operator<< (std::ostream &os, const Covering &ri)

0.15.126.1 Detailed Description

Semantics: The space is divided into a number of intervals: (-oo,a)[a,a](a,b)[b,b](b,oo) A bit is set if the constraints refutes the corresponding interval.

0.15.126.2 Constructor & Destructor Documentation

0.15.126.2.1 Covering()

```
smtrat::mcsat::arithmetic::Covering::Covering (
    std::size_t intervals ) [inline]
```

0.15.126.3 Member Function Documentation

0.15.126.3.1 add()

```
void smtrat::mcsat::arithmetic::Covering::add (
    const FormulaT & c,
    const carl::Bitset & b ) [inline]
```

0.15.126.3.2 buildConflictingCore()

```
void smtrat::mcsat::arithmetic::Covering::buildConflicting↩
Core (
    std::vector< FormulaT > & core ) const [inline]
```

0.15.126.3.3 conflicts()

```
bool smtrat::mcsat::arithmetic::Covering::conflicts ( ) const [inline]
```

0.15.126.3.4 satisfyingInterval()

```
std::size_t smtrat::mcsat::arithmetic::Covering::satisfying↩
Interval ( ) const [inline]
```

0.15.126.3.5 satisfyingSamples()

```
const auto& smtrat::mcsat::arithmetic::Covering::satisfying↩
Samples ( ) const [inline]
```

0.15.126.4 Friends And Related Function Documentation

0.15.126.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const Covering & ri ) [friend]
```

0.15.127 smtrat::cadcells::representation::covering< CoveringHeuristic::BIGGEST_CELL_COVERING > Struct Reference

```
#include <heuristics_covering.h>
```

Static Public Member Functions

- template<typename T >
  static std::optional< datastructures::CoveringRepresentation< T > > compute (const std::vector< datastructures::SampledDerivationRef< T >> &derivs)

0.15.127.1 Member Function Documentation

0.15.127.1.1 compute()

```
template<typename T >
static std::optional<datastructures::CoveringRepresentation<T> > smtrat::cadcells::representation::covering<
CoveringHeuristic::BIGGEST_CELL_COVERING >::compute (
            const std::vector< datastructures::SampledDerivationRef< T >> & derivs )  [inline],
[static]
```

0.15.128 smtrat::cadcells::representation::covering< CoveringHeuristic::BIGGEST_CELL_COVERING_EW > Struct Reference

```
#include <heuristics_covering.h>
```

Static Public Member Functions

- template<typename T >
  static std::optional< datastructures::CoveringRepresentation< T > > compute (const std::vector< datastructures::SampledDerivationRef< T >> &derivs)

0.15.128.1 Member Function Documentation

0.15.128.1.1 compute()

```
template<typename T >
static std::optional<datastructures::CoveringRepresentation<T> > smtrat::cadcells::representation::covering<
CoveringHeuristic::BIGGEST_CELL_COVERING_EW >::compute (
            const std::vector< datastructures::SampledDerivationRef< T >> & derivs )  [inline],
[static]
```

0.15.129 smtrat::cadcells::representation::covering< CoveringHeuristic::BIGGEST_CELL_COVERING_MIN_TDEG > Struct Reference

```
#include <heuristics_covering.h>
```

Static Public Member Functions

- template<typename T >
  static std::optional< datastructures::CoveringRepresentation< T > > compute (const std::vector< datastructures::SampledDerivationRef< T >> &derivs)

0.15.129.1 Member Function Documentation

0.15.129.1.1 compute()

```
template<typename T >
static std::optional<datastructures::CoveringRepresentation<T> > smtrat::cadcells::representation::covering<
CoveringHeuristic::BIGGEST_CELL_COVERING_MIN_TDEG >::compute (
            const std::vector< datastructures::SampledDerivationRef< T >> & derivs )  [inline],
[static]
```

0.15.130 smtrat::cadcells::representation::covering< CoveringHeuristic::CHAIN_COVERING > Struct Reference

```
#include <heuristics_covering.h>
```

Static Public Member Functions

- template<typename T >
  static std::optional< datastructures::CoveringRepresentation< T > > compute (const std::vector< datastructures::SampledDerivationRef< T >> &derivs)

0.15.130.1 Member Function Documentation

0.15.130.1.1 compute()

```
template<typename T >
static std::optional<datastructures::CoveringRepresentation<T> > smtrat::cadcells::representation::covering<
CoveringHeuristic::CHAIN_COVERING >::compute (
            const std::vector< datastructures::SampledDerivationRef< T >> & derivs )  [inline],
[static]
```

0.15.131 smtrat::cadcells::datastructures::CoveringDescription Class Reference

Describes a covering of the real line by SymbolicIntervals (given an appropriate sample).

```
#include <roots.h>
```

Public Member Functions

- void add (const SymbolicInterval &c)

  Add a SymbolicInterval to the covering.
- const auto & cells () const

0.15.131.1 Detailed Description

Describes a covering of the real line by SymbolicIntervals (given an appropriate sample).

0.15.131.2 Member Function Documentation

0.15.131.2.1 add()

```
void smtrat::cadcells::datastructures::CoveringDescription::add (
            const SymbolicInterval & c )  [inline]
```

Add a SymbolicInterval to the covering.
The added cell needs to be the rightmost cell of the already added cells and not be contained in any of these cells (or vice versa).

- The first cell needs to have -oo as left bound.
- The last cell needs to have oo as right bound.
- All cells need to cover the real line under an appropriate sample.
- Evaluated under an appropriate sample, the left bound of the added cell c is strictly greater than the left bounds of already added cells (considering also "strictness" of the bounds).

- The right bound of the added cell c needs to be greater than all right bounds of already added cells (considering also "strictness" of the bounds).

0.15.131.2.2 cells() `const auto& smtrat::cadcells::datastructures::CoveringDescription::cells ( ) const [inline]`

0.15.132 smtrat::CoveringNGModule< Settings > Class Template Reference

`#include <CoveringNGModule.h>`

Public Types

- using SettingsType = Settings
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- CoveringNGModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=nullptr)
- ~CoveringNGModule ()
- bool informCore (const FormulaT &_constraint)
- bool addCore (ModuleInput::const_iterator _subformula)
- void removeCore (ModuleInput::const_iterator _subformula)
- 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
- 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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.
- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)
- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.

- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

  Copies the infeasible subsets of the passed formula.
- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const

- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.

- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.132.1 Member Typedef Documentation

0.15.132.1.1 SettingsType

```
template<typename Settings >
using smtrat::CoveringNGModule< Settings >::SettingsType = Settings
```

0.15.132.2 Member Enumeration Documentation

0.15.132.2.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

0.15.132.3 Constructor & Destructor Documentation

0.15.132.3.1 CoveringNGModule()

```
template<typename Settings >
smtrat::CoveringNGModule< Settings >::CoveringNGModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

0.15.132.3.2 ~CoveringNGModule()

```
template<typename Settings >
smtrat::CoveringNGModule< Settings >::~CoveringNGModule ( )
```

0.15.132.4 Member Function Documentation

0.15.132.4.1 add()

```
bool smtrat::Module::add (
            ModuleInput::const_iterator _subformula ) [inherited]
```

The module has to take the given sub-formula of the received formula into account.

Parameters

_subformula	The sub-formula to take additionally into account.

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

0.15.132.4.2 addConstraintToInform()

```
void smtrat::Module::addConstraintToInform (
    const FormulaT & _constraint ) [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

0.15.132.4.3 addCore() [1/2]

```
virtual bool smtrat::Module::addCore (
    [[maybe_unused] ] 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.

0.15.132.4.4 addCore() [2/2]

```
template<typename Settings >
bool smtrat::CoveringNGModule< Settings >::addCore (
    ModuleInput::const_iterator _subformula )
```

0.15.132.4.5 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula
(
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.132.4.6 addLemma()

```
void smtrat::Module::addLemma (
    const FormulaT & _lemma,
    const LemmaType & _lt = LemmaType::NORMAL,
    const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline],
[inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.132.4.7 addOrigin()

```
void smtrat::Module::addOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.132.4.8 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.132.4.9 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.132.4.10 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.132.4.11 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.

0.15.132.4.12 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns

All satisfying assignments, if existent.

0.15.132.4.13 anAnswerFound()

```
bool smtrat::Module::anAnswerFound ( ) const [inline], [protected],
[inherited]
```

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.132.4.14 answerFound()

```
const smtrat::Conditionals& smtrat::Module::answerFound ( ) const
[inline], [inherited]
```

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.132.4.15 backendsModel()

```
const Model & smtrat::Module::backendsModel ( ) const [protected],
[inherited]
```

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.132.4.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.132.4.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.132.4.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.132.4.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.132.4.20 check()

```
Answer smtrat::Module::check (
    bool _final = false,
    bool _full = true,
    carl::Variable _objective = carl::Variable::NO_VARIABLE ) [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.132.4.21 checkCore()

```
template<typename Settings >
Answer smtrat::CoveringNGModule< Settings >::checkCore ( ) [virtual]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.132.4.22 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.

0.15.132.4.23 checkModel()

```
unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]
```

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.132.4.24 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.132.4.25 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

0.15.132.4.26 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

0.15.132.4.27 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

0.15.132.4.28 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.132.4.29 collectOrigins() [1/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

0.15.132.4.30 collectOrigins() [2/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inherited]`

Collects the formulas in the given formula, which are part of the received formula.
If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.132.4.31 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

0.15.132.4.32 constraintsToInform() `const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns

The constraints which the backends must be informed about.

0.15.132.4.33 currentlySatisfied()

`virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.132.4.34 currentlySatisfiedByBackend()

`unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.132.4.35 deinform()

`void smtrat::Module::deinform ( const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.132.4.36 deinformCore()

`virtual void smtrat::Module::deinformCore ( [[maybe_unused] ] 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.

0.15.132.4.37 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

0.15.132.4.38 eraseSubformulaFromPassedFormula()

`ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula ( ModuleInput::iterator _subformula, bool _ignoreOrigins = false ) [protected], [virtual], [inherited]`

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_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 >.

0.15.132.4.39 excludeNotReceivedVariablesFromModel()

`void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

0.15.132.4.40 findBestOrigin()

`std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin ( const std::vector< FormulaT > & _origins ) const [protected], [inherited]`

Parameters

_origins	

Returns

0.15.132.4.41 firstSubformulaToPass()

`ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.132.4.42 firstUncheckedReceivedSubformula()

`ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.132.4.43 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.132.4.44 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasible↩Subset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.132.4.45 getBackendsAllModels()

`void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.132.4.46 getBackendsModel()

`void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.132.4.47 getInfeasibleSubsets() [1/2]

`void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.132.4.48 getInfeasibleSubsets() [2/2]

`std::vector< FormulaSetT > smtrat::Module::get↩InfeasibleSubsets ( 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.

0.15.132.4.49 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module↩::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.132.4.50 getModelEqualities()

`std::list< std::vector< carl::Variable > > smtrat::Module↩::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.132.4.51 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
const FormulaT & _formula,
FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.132.4.52 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
const FormulaT & _formula,
FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.132.4.53 getOrigins() [3/3]

```
const FormulaT& smtrat::Module::getOrigins (
ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]
```

Gets the origins of the passed formula at the given position.

Parameters

_formula	The position of a formula in the passed formulas.

Returns

The origins of the passed formula at the given position.

0.15.132.4.54 getReceivedFormulaSimplified()

```
std::pair< bool, FormulaT > smtrat::Module::get↩
ReceivedFormulaSimplified ( ) [virtual], [inherited]
```

Returns

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's reveived formula, if the Boolean is true.

Reimplemented in smtrat::PModule.

0.15.132.4.55 hasLemmas()

```
bool smtrat::Module::hasLemmas ( ) [inline], [inherited]
```

Checks whether this module or any of its backends provides any lemmas.

0.15.132.4.56 hasValidInfeasibleSubset()

```
bool smtrat::Module::hasValidInfeasibleSubset ( ) const
[inherited]
```

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.132.4.57 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.132.4.58 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasible↩ Subsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.132.4.59 inform() `bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.
It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

Parameters

_constraint	The constraint to inform about.

Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

0.15.132.4.60 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.

0.15.132.4.61 informCore() [1/2] `virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.132.4.62 informCore() [2/2]

```
template<typename Settings >
bool smtrat::CoveringNGModule< Settings >::informCore (
            const FormulaT & _constraint )
```

0.15.132.4.63 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.

0.15.132.4.64 init()

```
void smtrat::Module::init ( ) [virtual], [inherited]
```

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not be called twice and only before the first runBackends call.
Reimplemented in smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntBlastModule< Settings >, smtrat::FPPModule< Settings >, smtrat::CurryModule< Settings >, and smtrat::BVModule< Settings >.

0.15.132.4.65 is_minimizing()

```
bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]
```

0.15.132.4.66 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.

0.15.132.4.67 isPreprocessor()

```
bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]
```

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.132.4.68 lemmas()

```
const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline],
[inherited]
```

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.132.4.69 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.

0.15.132.4.70 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.132.4.71 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.

0.15.132.4.72 moduleName() `virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]`

Returns

The name of the given module type as name.

Reimplemented in smtrat::VSModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::STropModule< Settings >, smtrat::SplitSOSModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntEqModule< Settings >, smtrat::IntBlastModule< Settings >, smtrat::IncWidthModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::GBModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::CurryModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::BVModule< Settings >, and smtrat::BEModule< Settings >.

0.15.132.4.73 objective() `carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

0.15.132.4.74 originInReceivedFormula() `bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.132.4.75 passedFormulaBegin() `ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.132.4.76 passedFormulaEnd() `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.132.4.77 pPassedFormula() `const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.132.4.78 pReceivedFormula() `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.132.4.79 print() `void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.132.4.80 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.

0.15.132.4.81 printInfeasibleSubsets() `void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.132.4.82 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.

0.15.132.4.83 printPassedFormula()

`void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.132.4.84 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.132.4.85 probablyLooping()

`bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.132.4.86 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.132.4.87 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩
InfeasibleSubset (
            ModuleInput::const_iterator _subformula )  [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

0.15.132.4.88 receivedVariable()

```
bool smtrat::Module::receivedVariable (
            carl::Variable::Arg _var ) const  [inline], [inherited]
```

0.15.132.4.89 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.

0.15.132.4.90 removeCore() [1/2]

```
virtual void smtrat::Module::removeCore (
            [[maybe_unused] ] 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.

0.15.132.4.91 removeCore() [2/2]

```
template<typename Settings >
void smtrat::CoveringNGModule< Settings >::removeCore (
            ModuleInput::const_iterator _subformula )
```

0.15.132.4.92 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origin (
            ModuleInput::iterator _formula,
            const FormulaT & _origin )  [inline], [protected], [inherited]
```

0.15.132.4.93 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins )  [inline], [protected],
[inherited]
```

0.15.132.4.94 rPassedFormula() `const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]`

Returns

A reference to the formula passed to the backends of this module.

0.15.132.4.95 rReceivedFormula() `const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]`

Returns

A reference to the formula passed to this module.

0.15.132.4.96 runBackends() [1/2] `virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]`

Reimplemented in smtrat::PModule.

0.15.132.4.97 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.

0.15.132.4.98 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.

0.15.132.4.99 setThreadPriority()

```
void smtrat::Module::setThreadPriority (
    thread_priority _threadPriority ) [inline], [inherited]
```

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.132.4.100 solverState() `Answer smtrat::Module::solverState ( ) const [inline], [inherited]`

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.132.4.101 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 !=.

0.15.132.4.102 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.132.4.103 updateAllModels() `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.132.4.104 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.132.4.105 updateModel() `template<typename Settings > void smtrat::CoveringNGModule< Settings >::updateModel ( ) const [virtual]`

Updates the model, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented from smtrat::Module.

0.15.132.4.106 usedBackends() `const std::vector<Module*>& smtrat::Module::usedBackends ( ) const [inline], [inherited]`

Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.132.5 Field Documentation

0.15.132.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`
The backends of this module which have been used.

0.15.132.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`
Stores all satisfying assignments.

0.15.132.5.3 mBackendsFoundAnswer `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`
This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.132.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`
Stores the constraints which the backends must be informed about.

0.15.132.5.5 mFinalCheck `bool smtrat::Module::mFinalCheck [protected], [inherited]`
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.132.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`
The beginning of the cyclic buffer storing the last branches.

0.15.132.5.7 mFirstSubformulaToPass `ModuleInput::iterator smtrat::Module::mFirstSubformulaTo↩Pass [protected], [inherited]`
Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.132.5.8 mFirstUncheckedReceivedSubformula `ModuleInput::const_iterator smtrat::Module↩::mFirstUncheckedReceivedSubformula [protected], [inherited]`
Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.132.5.9 mFoundAnswer `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.132.5.10 mFullCheck `bool smtrat::Module::mFullCheck [protected], [inherited]`
false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.132.5.11 mInfeasibleSubsets `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`
Stores the infeasible subsets.

0.15.132.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.132.5.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std↩::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.132.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.132.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.132.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.132.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.132.5.18 mObjectiveVariable

`carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.132.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplitting↩Variables [static], [inherited]`

Reusable splitting variables.

0.15.132.5.20 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.132.5.21 mSmallerMusesCheckCounter

`unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.132.5.22 mSolverState

`std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.132.5.23 mTheoryPropagations

`std::vector<TheoryPropagation> smtrat::Module::mTheory↩Propagations [protected], [inherited]`

0.15.132.5.24 mUsedBackends `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.132.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.133 smtrat::cadcells::datastructures::CoveringRepresentation< P > Struct Template Reference

Represents a covering over a cell.

`#include <representation.h>`

Public Member Functions

- CoveringDescription get_covering () const

  Returns a descriptions of the covering.
- std::vector< SampledDerivationRef< P > > sampled_derivations ()

  Returns the derivations.
- bool is_valid () const

  Checks whether this represents a proper non-redundant covering.

Data Fields

- std::vector< CellRepresentation< P > > cells

  Cells of the covering in increasing order and no cell is contained in another cell.
- IndexedRootOrdering ordering

  An ordering on the roots for the cell boundaries mainting the covering.

0.15.133.1 Detailed Description

template<typename P>
struct smtrat::cadcells::datastructures::CoveringRepresentation< P >

Represents a covering over a cell.
The cells forming the covering are in increasing order (ordered by lower bound) and no cell is contained in another cell.

0.15.133.2 Member Function Documentation

0.15.133.2.1 get_covering() `template<typename P > CoveringDescription smtrat::cadcells::datastructures::CoveringRepresentation< P >::get_covering ( ) const [inline]`

Returns a descriptions of the covering.

0.15.133.2.2 is_valid() `template<typename P > bool smtrat::cadcells::datastructures::CoveringRepresentation< P >::is_valid ( ) const [inline]`

Checks whether this represents a proper non-redundant covering.

0.15.133.2.3 sampled_derivations() `template<typename P >` `std::vector<SampledDerivationRef<P> > smtrat::cadcells::datastructures::CoveringRepresentation< P >::sampled_derivations ( ) [inline]`

Returns the derivations.

0.15.133.3 Field Documentation

0.15.133.3.1 cells `template<typename P >` `std::vector<CellRepresentation<P> > smtrat::cadcells::datastructures::CoveringRepresentation< P >::cells`

Cells of the covering in increasing order and no cell is contained in another cell.

0.15.133.3.2 ordering `template<typename P >` `IndexedRootOrdering smtrat::cadcells::datastructures::CoveringRepresentation< P >::ordering`

An ordering on the roots for the cell boundaries mainting the covering.

0.15.134 smtrat::covering_ng::CoveringResult< op > Struct Template Reference

`#include <types.h>`

Data Structures

- struct NONE

Public Types

- enum Status { SAT , UNSAT , FAILED_PROJECTION , FAILED }

Public Member Functions

- CoveringResult ()
- CoveringResult (Status s)
- CoveringResult (std::vector< Interval< op >> &c)
- CoveringResult (const cadcells::Assignment &c)
- bool is_failed ()
- bool is_failed_projection ()
- bool is_sat () const
- bool is_unsat () const
- const auto & sample () const
- const auto & intervals () const

Data Fields

- Status status
- std::variant< std::vector< Interval< op > >, cadcells::Assignment, NONE > content

0.15.134.1 Member Enumeration Documentation

0.15.134.1.1 Status `template<cadcells::operators::op op>` `enum smtrat::covering_ng::CoveringResult::Status`

Enumerator

SAT	
UNSAT	
FAILED_PROJECTION	
FAILED	

0.15.134.2 Constructor & Destructor Documentation

0.15.134.2.1 CoveringResult() [1/4]

```
template<cadcells::operators::op op>
smtrat::covering_ng::CoveringResult< op >::CoveringResult ( ) [inline]
```

0.15.134.2.2 CoveringResult() [2/4]

```
template<cadcells::operators::op op>
smtrat::covering_ng::CoveringResult< op >::CoveringResult (
            Status s ) [inline]
```

0.15.134.2.3 CoveringResult() [3/4]

```
template<cadcells::operators::op op>
smtrat::covering_ng::CoveringResult< op >::CoveringResult (
            std::vector< Interval< op >> & c ) [inline]
```

0.15.134.2.4 CoveringResult() [4/4]

```
template<cadcells::operators::op op>
smtrat::covering_ng::CoveringResult< op >::CoveringResult (
            const cadcells::Assignment & c ) [inline]
```

0.15.134.3 Member Function Documentation

0.15.134.3.1 intervals()

```
template<cadcells::operators::op op>
const auto& smtrat::covering_ng::CoveringResult< op >::intervals ( ) const [inline]
```

0.15.134.3.2 is_failed()

```
template<cadcells::operators::op op>
bool smtrat::covering_ng::CoveringResult< op >::is_failed ( ) [inline]
```

0.15.134.3.3 is_failed_projection()

```
template<cadcells::operators::op op>
bool smtrat::covering_ng::CoveringResult< op >::is_failed_projection ( ) [inline]
```

0.15.134.3.4 is_sat()

```
template<cadcells::operators::op op>
bool smtrat::covering_ng::CoveringResult< op >::is_sat ( ) const [inline]
```

0.15.134.3.5 is_unsat()

```
template<cadcells::operators::op op>
bool smtrat::covering_ng::CoveringResult< op >::is_unsat ( ) const [inline]
```

0.15.134.3.6 sample()

```
template<cadcells::operators::op op>
const auto& smtrat::covering_ng::CoveringResult< op >::sample ( ) const [inline]
```

0.15.134.4 Field Documentation

0.15.134.4.1 content `template<cadcells::operators::op op>`
`std::variant<std::vector<Interval<op> >, cadcells::Assignment, NONE> smtrat::covering_ng::CoveringResult< op >::content`

0.15.134.4.2 status `template<cadcells::operators::op op>`
`Status smtrat::covering_ng::CoveringResult< op >::status`

0.15.135 smtrat::mcsat::onecellcad::recursive::CoverNullification Struct Reference

`#include <Explanation.h>`

Static Public Attributes

- static constexpr bool cover_nullification = true

0.15.135.1 Field Documentation

0.15.135.1.1 cover_nullification `constexpr bool smtrat::mcsat::onecellcad::recursive::Cover↩ Nullification::cover_nullification = true [static], [constexpr]`

0.15.136 smtrat::CSplitModule< Settings > Class Template Reference

`#include <CSplitModule.h>`

Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- CSplitModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=nullptr)
- bool addCore (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- void removeCore (ModuleInput::const_iterator _subformula)

  Removes the subformula of the received formula at the given position to the considered ones of this module.
- void updateModel () const

  Updates the current assignment into the model.
- Answer checkCore ()

  Checks the received formula for consistency.
- bool inform (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void deinform (const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual void init ()

  Informs all backends about the so far encountered constraints, which have not yet been communicated.
- bool add (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.

- virtual Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_↩ VARIABLE)

  Checks the received formula for consistency.
- virtual void remove (ModuleInput::const_iterator _subformula)

  Removes everything related to the given sub-formula of the received formula.
- virtual void updateAllModels ()

  Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
- virtual std::list< std::vector< carl::Variable > > getModelEqualities () const

  Partition the variables from the current model into equivalence classes according to their assigned value.
- unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const
- virtual unsigned currentlySatisfied (const FormulaT &) const
- bool receivedVariable (carl::Variable::Arg _var) const
- Answer solverState () const
- std::size_t id () const
- void setId (std::size_t _id)

  Sets this 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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.

- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)
- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

  Copies the infeasible subsets of the passed formula.
- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

Merges the two vectors of sets into the first one.

- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.136.1 Member Typedef Documentation

0.15.136.1.1 SettingsType `template<typename Settings >`

```
typedef Settings smtrat::CSplitModule< Settings >::SettingsType
```

0.15.136.2 Member Enumeration Documentation

0.15.136.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

0.15.136.3 Constructor & Destructor Documentation

0.15.136.3.1 CSplitModule() `template<typename Settings >`

```
smtrat::CSplitModule< Settings >::CSplitModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

0.15.136.4 Member Function Documentation

0.15.136.4.1 add() `bool smtrat::Module::add ( ModuleInput::const_iterator _subformula ) [inherited]`

The module has to take the given sub-formula of the received formula into account.

Parameters

_subformula	The sub-formula to take additionally into account.

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

0.15.136.4.2 addConstraintToInform() `void smtrat::Module::addConstraintToInform ( const FormulaT & _constraint ) [protected], [virtual], [inherited]`

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

0.15.136.4.3 addCore() [1/2] `virtual bool smtrat::Module::addCore ( [[maybe_unused] ] 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.

0.15.136.4.4 addCore() [2/2] `template<typename Settings > bool smtrat::CSplitModule< Settings >::addCore ( 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.

0.15.136.4.5 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.136.4.6 addLemma()

```
void smtrat::Module::addLemma (
    const FormulaT & _lemma,
    const LemmaType & _lt = LemmaType::NORMAL,
    const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline],
[inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.136.4.7 addOrigin()

```
void smtrat::Module::addOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.136.4.8 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.136.4.9 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula )  [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.136.4.10 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula,
            const FormulaT & _origin )  [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.136.4.11 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.

0.15.136.4.12 allModels()

`const std::vector<Model>& smtrat::Module::allModels ( ) const [inline], [inherited]`

Returns

All satisfying assignments, if existent.

0.15.136.4.13 anAnswerFound()

`bool smtrat::Module::anAnswerFound ( ) const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.136.4.14 answerFound()

`const smtrat::Conditionals& smtrat::Module::answerFound ( ) const [inline], [inherited]`

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.136.4.15 backendsModel()

`const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.136.4.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.136.4.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.136.4.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
```

```
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline],
[protected], [inherited]
```

0.15.136.4.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
const Poly & _polynomial,
bool _integral,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.136.4.20 check()

```
Answer smtrat::Module::check (
bool _final = false,
bool _full = true,
carl::Variable _objective = carl::Variable::NO_VARIABLE ) [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.136.4.21 checkCore() `template<typename Settings > Answer smtrat::CSplitModule< Settings >::checkCore ( ) [virtual]`

Checks the received formula for consistency.

Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; UNKNOWN, otherwise.

Reimplemented from smtrat::Module.

0.15.136.4.22 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.

0.15.136.4.23 checkModel() `unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]`

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.136.4.24 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.136.4.25 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

0.15.136.4.26 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

0.15.136.4.27 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

0.15.136.4.28 clearPassedFormula()

`void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.136.4.29 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

0.15.136.4.30 collectOrigins() [2/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.
If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.136.4.31 collectTheoryPropagations()

`void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

0.15.136.4.32 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns

The constraints which the backends must be informed about.

0.15.136.4.33 currentlySatisfied()

```
virtual unsigned smtrat::Module::currentlySatisfied (
    const FormulaT & ) const [inline], [virtual], [inherited]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.136.4.34 currentlySatisfiedByBackend()

```
unsigned smtrat::Module::currentlySatisfiedByBackend (
    const FormulaT & _formula ) const [inherited]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.136.4.35 deinform()

```
void smtrat::Module::deinform (
    const FormulaT & _constraint ) [inherited]
```

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.136.4.36 deinformCore()

```
virtual void smtrat::Module::deinformCore (
    [[maybe_unused] ] 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.

0.15.136.4.37 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

0.15.136.4.38 eraseSubformulaFromPassedFormula()

```
ModuleInput::iterator smtrat::Module::erase↩
SubformulaFromPassedFormula (
    ModuleInput::iterator _subformula,
    bool _ignoreOrigins = false ) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_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 >.

0.15.136.4.39 excludeNotReceivedVariablesFromModel() `void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

0.15.136.4.40 findBestOrigin() `std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin ( const std::vector< FormulaT > & _origins ) const [protected], [inherited]`

Parameters

_origins	

Returns

0.15.136.4.41 firstSubformulaToPass() `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.136.4.42 firstUncheckedReceivedSubformula() `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.136.4.43 freeSplittingVariable() `static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.136.4.44 generateTrivialInfeasibleSubset() `void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.136.4.45 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.136.4.46 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.136.4.47 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.136.4.48 getInfeasibleSubsets() [2/2]

```
std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets (
    const Module & _backend ) const [protected], [inherited]
```

Get the infeasible subsets the given backend provides.
Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

Parameters

_backend	The backend from which to obtain the infeasible subsets.

Returns

The infeasible subsets the given backend provides.

0.15.136.4.49 getInformationRelevantFormulas()

```
const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]
```

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.136.4.50 getModelEqualities()

```
std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]
```

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.136.4.51 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.136.4.52 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.136.4.53 getOrigins() [3/3] const FormulaT& smtrat::Module::getOrigins (ModuleInput::const_iterator _formula) const [inline], [protected], [inherited]

Gets the origins of the passed formula at the given position.

Parameters

_formula	The position of a formula in the passed formulas.

Returns

The origins of the passed formula at the given position.

0.15.136.4.54 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.

0.15.136.4.55 hasLemmas() bool smtrat::Module::hasLemmas ( ) [inline], [inherited]

Checks whether this module or any of its backends provides any lemmas.

0.15.136.4.56 hasValidInfeasibleSubset() bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.136.4.57 id() std::size_t smtrat::Module::id ( ) const [inline], [inherited]

Returns

A unique ID to identify this module instance.

0.15.136.4.58 infeasibleSubsets() const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets ( ) const [inline], [inherited]

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.136.4.59 inform() bool smtrat::Module::inform (const FormulaT & _constraint) [inherited]

Informs the module about the given constraint.
It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

Parameters

_constraint	The constraint to inform about.

Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

0.15.136.4.60 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.

0.15.136.4.61 informCore()

`virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.136.4.62 informedConstraints()

`const carl::FastSet<FormulaT>& smtrat::Module::informedConstraints ( ) const [inline], [inherited]`

Returns

The position of the first constraint of which no backend has been informed about.

0.15.136.4.63 init()

`void smtrat::Module::init ( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not be called twice and only before the first runBackends call.
Reimplemented in smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntBlastModule< Settings >, smtrat::FPPModule< Settings >, smtrat::CurryModule< Settings >, and smtrat::BVModule< Settings >.

0.15.136.4.64 is_minimizing()

`bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.136.4.65 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.

0.15.136.4.66 isPreprocessor()

`bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.136.4.67 lemmas()

`const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.136.4.68 merge()

`std::vector< FormulaT > smtrat::Module::merge ( const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.
({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

Parameters

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns

The vector being the two given vectors merged.

0.15.136.4.69 model()

`const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.136.4.70 modelsDisjoint()

`bool smtrat::Module::modelsDisjoint ( const Model & _modelA, const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

Parameters

_modelA	The first model to check for.
_modelB	The second model to check for.

Returns

true, if there is no variable assigned by both models; false, otherwise.

0.15.136.4.71 moduleName()

`template<typename Settings > std::string smtrat::CSplitModule< Settings >::moduleName ( ) const [inline], [virtual]`

Returns

The name of the given module type as name.

Reimplemented from smtrat::Module.

0.15.136.4.72 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

0.15.136.4.73 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.136.4.74 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.136.4.75 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.136.4.76 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.136.4.77 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.136.4.78 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.136.4.79 printAllModels()

`void smtrat::Module::printAllModels ( std::ostream & _out = std::cout ) [inherited]`

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.136.4.80 printInfeasibleSubsets()

`void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.136.4.81 printModel()

`void smtrat::Module::printModel ( std::ostream & _out = std::cout ) const [inherited]`

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.136.4.82 printPassedFormula()

`void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.136.4.83 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.136.4.84 probablyLooping()

```
bool smtrat::Module::probablyLooping (
    const typename Poly::PolyType & _branchingPolynomial,
    const Rational & _branchingValue ) const [protected], [inherited]
```

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.136.4.85 receivedFormulaChecked()

```
void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]
```

Notifies that the received formulas has been checked.

0.15.136.4.86 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩InfeasibleSubset (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

0.15.136.4.87 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const [inline], [inherited]
```

0.15.136.4.88 remove()

```
void smtrat::Module::remove (
    ModuleInput::const_iterator _subformula ) [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.
However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

Parameters

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

0.15.136.4.89 removeCore() [1/2]

```
virtual void smtrat::Module::removeCore (
    [[maybe_unused] ] 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.

0.15.136.4.90 removeCore() [2/2]

```
template<typename Settings >
void smtrat::CSplitModule< Settings >::removeCore (
            ModuleInput::const_iterator _subformula )
```

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.

0.15.136.4.91 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origin (
            ModuleInput::iterator _formula,
            const FormulaT & _origin )  [inline], [protected], [inherited]
```

0.15.136.4.92 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins )  [inline], [protected],
[inherited]
```

0.15.136.4.93 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.136.4.94 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to this module.

0.15.136.4.95 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( )  [inline],
[protected], [virtual], [inherited]
```

Reimplemented in smtrat::PModule.

0.15.136.4.96 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.

0.15.136.4.97 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.

0.15.136.4.98 setThreadPriority()

`void smtrat::Module::setThreadPriority ( thread_priority _threadPriority ) [inline], [inherited]`

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.136.4.99 solverState()

`Answer smtrat::Module::solverState ( ) const [inline], [inherited]`

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.136.4.100 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 !=.

0.15.136.4.101 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.136.4.102 updateAllModels() `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.136.4.103 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.136.4.104 updateModel() `template<typename Settings > void smtrat::CSplitModule< Settings >::updateModel ( ) const [virtual]`

Updates the current assignment into the model.
Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.
Reimplemented from smtrat::Module.

0.15.136.4.105 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).

0.15.136.5 Field Documentation

0.15.136.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.136.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.136.5.3 mBackendsFoundAnswer `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.136.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.136.5.5 mFinalCheck `bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.136.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.136.5.7 mFirstSubformulaToPass `ModuleInput::iterator smtrat::Module::mFirstSubformulaTo↩ Pass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.136.5.8 mFirstUncheckedReceivedSubformula `ModuleInput::const_iterator smtrat::Module↩ ::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.136.5.9 mFoundAnswer `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.136.5.10 mFullCheck `bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.136.5.11 mInfeasibleSubsets `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.136.5.12 mInformedConstraints `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.136.5.13 mLastBranches `std::vector< Branching > smtrat::Module::mLastBranches = std↩ ::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.136.5.14 mLemmas `std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.136.5.15 mModel `Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.136.5.16 mModelComputed `bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.136.5.17 mNumOfBranchVarsToStore `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.136.5.18 mObjectiveVariable `carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.136.5.19 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

0.15.136.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.136.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.136.5.22 mSolverState `std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.136.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

0.15.136.5.24 mUsedBackends `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.136.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.137 smtrat::CubeLIAModule< Settings > Class Template Reference

`#include <CubeLIAModule.h>`

Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- CubeLIAModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=nullptr)
- ~CubeLIAModule ()
- bool addCore (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- void removeCore (ModuleInput::const_iterator _subformula)

  Removes the subformula of the received formula at the given position to the considered ones of this module.
- void updateModel () const

  Updates the current assignment into the model.
- Answer checkCore ()

  Checks the received formula for consistency.
- bool inform (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void deinform (const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual void init ()

  Informs all backends about the so far encountered constraints, which have not yet been communicated.
- bool add (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- virtual Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_↩VARIABLE)

  Checks the received formula for consistency.
- virtual void remove (ModuleInput::const_iterator _subformula)

  Removes everything related to the given sub-formula of the received formula.
- virtual void updateAllModels ()

  Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
- virtual std::list< std::vector< carl::Variable > > getModelEqualities () const

  Partition the variables from the current model into equivalence classes according to their assigned value.
- unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const
- virtual unsigned currentlySatisfied (const FormulaT &) const
- bool receivedVariable (carl::Variable::Arg _var) const
- Answer solverState () const
- std::size_t id () const
- void setId (std::size_t _id)

  Sets this 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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.
- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)
- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.

- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

  Copies the infeasible subsets of the passed formula.
- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.137.1 Member Typedef Documentation

0.15.137.1.1 SettingsType

```
template<typename Settings >
typedef Settings smtrat::CubeLIAModule< Settings >::SettingsType
```

0.15.137.2 Member Enumeration Documentation

0.15.137.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

0.15.137.3 Constructor & Destructor Documentation

0.15.137.3.1 CubeLIAModule() `template<typename Settings >`

```
smtrat::CubeLIAModule< Settings >::CubeLIAModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

0.15.137.3.2 ~CubeLIAModule() `template<typename Settings >`

```
smtrat::CubeLIAModule< Settings >::~CubeLIAModule ( )
```

0.15.137.4 Member Function Documentation

0.15.137.4.1 add() `bool smtrat::Module::add (`

```
            ModuleInput::const_iterator _subformula )  [inherited]
```

The module has to take the given sub-formula of the received formula into account.

Parameters

_subformula	The sub-formula to take additionally into account.

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

0.15.137.4.2 addConstraintToInform() `void smtrat::Module::addConstraintToInform (`

```
            const FormulaT & _constraint )  [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

0.15.137.4.3 addCore() [1/2] `virtual bool smtrat::Module::addCore (`

```
[[maybe_unused] ] 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.

0.15.137.4.4 addCore() [2/2]

```
template<typename Settings >
bool smtrat::CubeLIAModule< Settings >::addCore (
            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.

0.15.137.4.5 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula
(
            const FormulaT & formula )  [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.137.4.6 addLemma()

```
void smtrat::Module::addLemma (
            const FormulaT & _lemma,
            const LemmaType & _lt = LemmaType::NORMAL,
            const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) )  [inline],
[inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.137.4.7 addOrigin()

```
void smtrat::Module::addOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.137.4.8 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.137.4.9 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.137.4.10 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.137.4.11 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.

0.15.137.4.12 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns

All satisfying assignments, if existent.

0.15.137.4.13 anAnswerFound()

```
bool smtrat::Module::anAnswerFound ( ) const [inline], [protected],
[inherited]
```

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.137.4.14 answerFound()

```
const smtrat::Conditionals& smtrat::Module::answerFound ( ) const
[inline], [inherited]
```

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.137.4.15 backendsModel()

```
const Model & smtrat::Module::backendsModel ( ) const [protected],
[inherited]
```

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.137.4.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.137.4.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.137.4.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.137.4.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.137.4.20 check()

```
Answer smtrat::Module::check (
    bool _final = false,
    bool _full = true,
    carl::Variable _objective = carl::Variable::NO_VARIABLE ) [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.137.4.21 checkCore()

```
template<typename 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.

0.15.137.4.22 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.

0.15.137.4.23 checkModel()

```
unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]
```

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.137.4.24 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.137.4.25 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

0.15.137.4.26 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

0.15.137.4.27 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

0.15.137.4.28 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.137.4.29 collectOrigins() [1/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

0.15.137.4.30 collectOrigins() [2/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inherited]`

Collects the formulas in the given formula, which are part of the received formula.
If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.137.4.31 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

0.15.137.4.32 constraintsToInform() `const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns

The constraints which the backends must be informed about.

0.15.137.4.33 currentlySatisfied()

`virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.137.4.34 currentlySatisfiedByBackend()

`unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.137.4.35 deinform()

`void smtrat::Module::deinform ( const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.137.4.36 deinformCore()

`virtual void smtrat::Module::deinformCore ( [[maybe_unused] ] 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.

0.15.137.4.37 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

0.15.137.4.38 eraseSubformulaFromPassedFormula()

```
ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (
            ModuleInput::iterator _subformula,
            bool _ignoreOrigins = false ) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_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 >.

0.15.137.4.39 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

0.15.137.4.40 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
            const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

0.15.137.4.41 firstSubformulaToPass()

```
ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]
```

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.137.4.42 firstUncheckedReceivedSubformula()

```
ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]
```

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.137.4.43 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.137.4.44 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasible↩ Subset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.137.4.45 getBackendsAllModels()

`void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.137.4.46 getBackendsModel()

`void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.137.4.47 getInfeasibleSubsets() [1/2]

`void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.137.4.48 getInfeasibleSubsets() [2/2]

`std::vector< FormulaSetT > smtrat::Module::get↩ InfeasibleSubsets ( 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.

0.15.137.4.49 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module↩ ::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.137.4.50 getModelEqualities()

`std::list< std::vector< carl::Variable > > smtrat::Module↩ ::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.137.4.51 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
const FormulaT & _formula,
FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.137.4.52 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
const FormulaT & _formula,
FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.137.4.53 getOrigins() [3/3]

```
const FormulaT& smtrat::Module::getOrigins (
ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]
```

Gets the origins of the passed formula at the given position.

Parameters

_formula	The position of a formula in the passed formulas.

Returns

The origins of the passed formula at the given position.

0.15.137.4.54 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.

0.15.137.4.55 hasLemmas()

```
bool smtrat::Module::hasLemmas ( ) [inline], [inherited]
```

Checks whether this module or any of its backends provides any lemmas.

0.15.137.4.56 hasValidInfeasibleSubset()

```
bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]
```

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.137.4.57 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.137.4.58 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasible↩ Subsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.137.4.59 inform() `bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.
It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

Parameters

_constraint	The constraint to inform about.

Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

0.15.137.4.60 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.

0.15.137.4.61 informCore() `virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.137.4.62 informedConstraints() `const carl::FastSet<FormulaT>& smtrat::Module::informedConstraints ( ) const [inline], [inherited]`

Returns

The position of the first constraint of which no backend has been informed about.

0.15.137.4.63 init() `void smtrat::Module::init ( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not be called twice and only before the first runBackends call.
Reimplemented in smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntBlastModule< Settings >, smtrat::FPPModule< Settings >, smtrat::CurryModule< Settings >, and smtrat::BVModule< Settings >.

0.15.137.4.64 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.137.4.65 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.

0.15.137.4.66 isPreprocessor() `bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.137.4.67 lemmas() `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.137.4.68 merge() `std::vector< FormulaT > smtrat::Module::merge ( const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.
({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

Parameters

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns

The vector being the two given vectors merged.

0.15.137.4.69 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.137.4.70 modelsDisjoint() `bool smtrat::Module::modelsDisjoint ( const Model & _modelA, const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

Parameters

_modelA	The first model to check for.
_modelB	The second model to check for.

Returns

true, if there is no variable assigned by both models; false, otherwise.

0.15.137.4.71 moduleName() `template<typename Settings > std::string smtrat::CubeLIAModule< Settings >::moduleName ( ) const [inline], [virtual]`

Returns

The name of the given module type as name.

Reimplemented from smtrat::Module.

0.15.137.4.72 objective() `carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

0.15.137.4.73 originInReceivedFormula() `bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.137.4.74 passedFormulaBegin() `ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.137.4.75 passedFormulaEnd() `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.137.4.76 pPassedFormula() `const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.137.4.77 pReceivedFormula() `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.137.4.78 print() `void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.137.4.79 printAllModels() `void smtrat::Module::printAllModels ( std::ostream & _out = std::cout ) [inherited]`

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.137.4.80 printInfeasibleSubsets() `void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.137.4.81 printModel() `void smtrat::Module::printModel ( std::ostream & _out = std::cout ) const [inherited]`

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.137.4.82 printPassedFormula()

`void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.137.4.83 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.137.4.84 probablyLooping()

`bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.137.4.85 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.137.4.86 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

0.15.137.4.87 receivedVariable()

`bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

0.15.137.4.88 remove()

`void smtrat::Module::remove ( ModuleInput::const_iterator _subformula ) [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.
However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

Parameters

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

0.15.137.4.89 removeCore() [1/2]

```
virtual void smtrat::Module::removeCore (
            [[maybe_unused] ] 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.

0.15.137.4.90 removeCore() [2/2]

```
template<typename Settings >
void smtrat::CubeLIAModule< Settings >::removeCore (
            ModuleInput::const_iterator _subformula )
```

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.

0.15.137.4.91 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origin (
            ModuleInput::iterator _formula,
            const FormulaT & _origin ) [inline], [protected], [inherited]
```

0.15.137.4.92 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

0.15.137.4.93 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.137.4.94 rReceivedFormula() `const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]`

Returns

A reference to the formula passed to this module.

0.15.137.4.95 runBackends() [1/2] `virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]`

Reimplemented in smtrat::PModule.

0.15.137.4.96 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.

0.15.137.4.97 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.

0.15.137.4.98 setThreadPriority()

```
void smtrat::Module::setThreadPriority (
    thread_priority _threadPriority ) [inline], [inherited]
```

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.137.4.99 solverState() `Answer smtrat::Module::solverState ( ) const [inline], [inherited]`

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.137.4.100 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 !=.

0.15.137.4.101 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.137.4.102 updateAllModels()

`void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.137.4.103 updateLemmas()

`void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.137.4.104 updateModel()

`template<typename Settings > void smtrat::CubeLIAModule< Settings >::updateModel ( ) const [virtual]`

Updates the current assignment into the model.
Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.
Reimplemented from smtrat::Module.

0.15.137.4.105 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).

0.15.137.5 Field Documentation

0.15.137.5.1 mAllBackends

`std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.137.5.2 mAllModels

`std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.137.5.3 mBackendsFoundAnswer

`std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.137.5.4 mConstraintsToInform

`carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.137.5.5 mFinalCheck

`bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.137.5.6 mFirstPosInLastBranches

`std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.137.5.7 mFirstSubformulaToPass

`ModuleInput::iterator smtrat::Module::mFirstSubformulaToPass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.137.5.8 mFirstUncheckedReceivedSubformula

`ModuleInput::const_iterator smtrat::Module::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.137.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.137.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.137.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.137.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.137.5.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.137.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.137.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.137.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.137.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.137.5.18 mObjectiveVariable

`carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.137.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]`

Reusable splitting variables.

0.15.137.5.20 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.137.5.21 mSmallerMusesCheckCounter

`unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.137.5.22 mSolverState

`std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.137.5.23 mTheoryPropagations

`std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]`

0.15.137.5.24 mUsedBackends `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.137.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.138 smtrat::CurryModule< Settings > Class Template Reference

`#include <CurryModule.h>`

Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- 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_↩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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.
- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_↩ ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

  Copies the infeasible subsets of the passed formula.
- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_↩ branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAs↩ Premise=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 >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.

- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.138.1 Member Typedef Documentation

0.15.138.1.1 SettingsType

```
template<typename Settings >
typedef Settings smtrat::CurryModule< Settings >::SettingsType
```

0.15.138.2 Member Enumeration Documentation

0.15.138.2.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

0.15.138.3 Constructor & Destructor Documentation

0.15.138.3.1 CurryModule()

```
template<typename Settings >
smtrat::CurryModule< Settings >::CurryModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

0.15.138.3.2 ~CurryModule()

```
template<typename Settings >
smtrat::CurryModule< Settings >::~CurryModule ( )
```

0.15.138.4 Member Function Documentation

0.15.138.4.1 add()

```
bool smtrat::Module::add (
            ModuleInput::const_iterator _subformula ) [inherited]
```

The module has to take the given sub-formula of the received formula into account.

Parameters

_subformula	The sub-formula to take additionally into account.

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

0.15.138.4.2 addConstraintToInform()

```
void smtrat::Module::addConstraintToInform (
    const FormulaT & _constraint ) [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

0.15.138.4.3 addCore() [1/2]

```
virtual bool smtrat::Module::addCore (
    [[maybe_unused] ] 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.

0.15.138.4.4 addCore() [2/2]

```
template<typename Settings >
bool smtrat::CurryModule< Settings >::addCore (
    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.

0.15.138.4.5 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula
(
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.138.4.6 addLemma()

```
void smtrat::Module::addLemma (
    const FormulaT & _lemma,
    const LemmaType & _lt = LemmaType::NORMAL,
    const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline], [inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.138.4.7 addOrigin()

```
void smtrat::Module::addOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.138.4.8 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::addReceivedSubformulaToPassedFormula (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.138.4.9 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.138.4.10 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula,
            const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.138.4.11 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.

0.15.138.4.12 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns

All satisfying assignments, if existent.

0.15.138.4.13 anAnswerFound()

`bool smtrat::Module::anAnswerFound ( ) const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.138.4.14 answerFound()

`const smtrat::Conditionals& smtrat::Module::answerFound ( ) const [inline], [inherited]`

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.138.4.15 backendsModel()

`const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.138.4.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.138.4.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.138.4.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.138.4.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
```

```
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false )  [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.138.4.20 check()

```
Answer smtrat::Module::check (
bool _final = false,
bool _full = true,
carl::Variable _objective = carl::Variable::NO_VARIABLE )  [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.138.4.21 checkCore()

```
template<typename 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.

0.15.138.4.22 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.

0.15.138.4.23 checkModel()

```
unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]
```

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.138.4.24 cleanModel()

```
void smtrat::Module::cleanModel ( ) const [inline], [protected],
[inherited]
```

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.138.4.25 clearLemmas()

```
void smtrat::Module::clearLemmas ( ) [inline], [inherited]
```

Deletes all yet found lemmas.

0.15.138.4.26 clearModel()

```
void smtrat::Module::clearModel ( ) const [inline], [protected],
[inherited]
```

Clears the assignment, if any was found.

0.15.138.4.27 clearModels()

```
void smtrat::Module::clearModels ( ) const [inline], [protected],
[inherited]
```

Clears all assignments, if any was found.

0.15.138.4.28 clearPassedFormula()

```
void smtrat::Module::clearPassedFormula ( ) [protected],
[inherited]
```

0.15.138.4.29 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
const FormulaT & _formula,
FormulaSetT & _origins ) const [inherited]
```

0.15.138.4.30 collectOrigins() [2/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.
If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.138.4.31 collectTheoryPropagations()

```
void smtrat::Module::collectTheoryPropagations ( ) [inherited]
```

0.15.138.4.32 constraintsToInform()

```
const carl::FastSet<FormulaT>& smtrat::Module::constraints↩
ToInform ( ) const [inline], [inherited]
```

Returns

The constraints which the backends must be informed about.

0.15.138.4.33 currentlySatisfied()

```
virtual unsigned smtrat::Module::currentlySatisfied (
    const FormulaT & ) const [inline], [virtual], [inherited]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.138.4.34 currentlySatisfiedByBackend()

```
unsigned smtrat::Module::currentlySatisfiedByBackend (
    const FormulaT & _formula ) const [inherited]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.138.4.35 deinform()

```
void smtrat::Module::deinform (
    const FormulaT & _constraint ) [inherited]
```

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.138.4.36 deinformCore()

`virtual void smtrat::Module::deinformCore ( [[maybe_unused] ] 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.

0.15.138.4.37 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

0.15.138.4.38 eraseSubformulaFromPassedFormula()

`ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula ( ModuleInput::iterator _subformula, bool _ignoreOrigins = false ) [protected], [virtual], [inherited]`

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_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 >.

0.15.138.4.39 excludeNotReceivedVariablesFromModel()

`void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

0.15.138.4.40 findBestOrigin()

`std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin ( const std::vector< FormulaT > & _origins ) const [protected], [inherited]`

Parameters

_origins	

Returns

0.15.138.4.41 firstSubformulaToPass() `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.138.4.42 firstUncheckedReceivedSubformula() `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.138.4.43 freeSplittingVariable() `static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.138.4.44 generateTrivialInfeasibleSubset() `void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.138.4.45 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.138.4.46 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.138.4.47 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.138.4.48 getInfeasibleSubsets() [2/2] `std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets ( const Module & _backend ) const [protected], [inherited]`

Get the infeasible subsets the given backend provides.
Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

Parameters

_backend	The backend from which to obtain the infeasible subsets.

Returns

The infeasible subsets the given backend provides.

0.15.138.4.49 getInformationRelevantFormulas() `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.138.4.50 getModelEqualities() `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.138.4.51 getOrigins() [1/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.138.4.52 getOrigins() [2/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.138.4.53 getOrigins() [3/3] `const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]`

Gets the origins of the passed formula at the given position.

Parameters

_formula	The position of a formula in the passed formulas.

Returns

The origins of the passed formula at the given position.

0.15.138.4.54 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.

0.15.138.4.55 hasLemmas() `bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

0.15.138.4.56 hasValidInfeasibleSubset() `bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]`

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.138.4.57 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.138.4.58 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.138.4.59 inform() `bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.
It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

Parameters

_constraint	The constraint to inform about.

Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

0.15.138.4.60 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.

0.15.138.4.61 informCore() [1/2]

`virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.138.4.62 informCore() [2/2]

`template<typename Settings > bool smtrat::CurryModule< Settings >::informCore ( 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.

0.15.138.4.63 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.

0.15.138.4.64 init() template<typename Settings > void smtrat::CurryModule< Settings >::init ( ) [virtual]

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not and will not be called more than once and only before the first runBackends call.
Reimplemented from smtrat::Module.

0.15.138.4.65 is_minimizing() bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]

0.15.138.4.66 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.

0.15.138.4.67 isPreprocessor() bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.138.4.68 lemmas() const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.138.4.69 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.

0.15.138.4.70 model() const Model& smtrat::Module::model ( ) const [inline], [inherited]

Returns

The assignment of the current satisfiable result, if existent.

0.15.138.4.71 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.

0.15.138.4.72 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.

0.15.138.4.73 objective()

```
carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]
```

0.15.138.4.74 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
    const FormulaT & _origin ) const [protected], [inherited]
```

0.15.138.4.75 passedFormulaBegin()

```
ModuleInput::iterator smtrat::Module::passedFormulaBegin (
) [inline], [protected], [inherited]
```

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.138.4.76 passedFormulaEnd()

```
ModuleInput::iterator smtrat::Module::passedFormulaEnd ( )
[inline], [protected], [inherited]
```

Returns

An iterator to the end of the passed formula.

0.15.138.4.77 pPassedFormula()

```
const ModuleInput* smtrat::Module::pPassedFormula ( ) const
[inline], [inherited]
```

Returns

A pointer to the formula passed to the backends of this module.

0.15.138.4.78 pReceivedFormula() `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.138.4.79 print() `void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.138.4.80 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.

0.15.138.4.81 printInfeasibleSubsets() `void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.138.4.82 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.

0.15.138.4.83 printPassedFormula() `void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.138.4.84 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.138.4.85 probablyLooping()

`bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.138.4.86 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.138.4.87 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

0.15.138.4.88 receivedVariable()

`bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

0.15.138.4.89 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.

0.15.138.4.90 removeCore() [1/2]

`virtual void smtrat::Module::removeCore ( [[maybe_unused] ] 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.

0.15.138.4.91 removeCore() [2/2]

`template<typename Settings > void smtrat::CurryModule< Settings >::removeCore ( ModuleInput::const_iterator _subformula )`

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.

0.15.138.4.92 removeOrigin()

`std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin ( ModuleInput::iterator _formula, const FormulaT & _origin ) [inline], [protected], [inherited]`

0.15.138.4.93 removeOrigins()

`std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins ( ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]`

0.15.138.4.94 rPassedFormula()

`const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]`

Returns

A reference to the formula passed to the backends of this module.

0.15.138.4.95 rReceivedFormula()

`const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]`

Returns

A reference to the formula passed to this module.

0.15.138.4.96 runBackends() [1/2]

`virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]`

Reimplemented in smtrat::PModule.

0.15.138.4.97 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.

0.15.138.4.98 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.

0.15.138.4.99 setThreadPriority()

```
void smtrat::Module::setThreadPriority (
    thread_priority _threadPriority ) [inline], [inherited]
```

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.138.4.100 solverState()

```
Answer smtrat::Module::solverState ( ) const [inline], [inherited]
```

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.138.4.101 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 !=.

0.15.138.4.102 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.138.4.103 updateAllModels() `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.138.4.104 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.138.4.105 updateModel() `template<typename Settings > void smtrat::CurryModule< Settings >::updateModel ( ) const [virtual]`

Updates the current assignment into the model.
Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.
Reimplemented from smtrat::Module.

0.15.138.4.106 usedBackends() `const std::vector<Module*>& smtrat::Module::usedBackends ( ) const [inline], [inherited]`

Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.138.5 Field Documentation

0.15.138.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.138.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.138.5.3 mBackendsFoundAnswer `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.138.5.4 mConstraintsToInform

`carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.138.5.5 mFinalCheck

`bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.138.5.6 mFirstPosInLastBranches

`std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.138.5.7 mFirstSubformulaToPass

`ModuleInput::iterator smtrat::Module::mFirstSubformulaTo↩ Pass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.138.5.8 mFirstUncheckedReceivedSubformula

`ModuleInput::const_iterator smtrat::Module↩ ::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.138.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.138.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.138.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.138.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.138.5.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std↩ ::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.138.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.138.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.138.5.16 mModelComputed `bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.138.5.17 mNumOfBranchVarsToStore `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.138.5.18 mObjectiveVariable `carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.138.5.19 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

0.15.138.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.138.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.138.5.22 mSolverState `std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.138.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

0.15.138.5.24 mUsedBackends `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.138.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.139 smtrat::CycleEnumerator< FHG, Collector > Struct Template Reference

This class encapsulates an algorithm for enumerating all cycles.

`#include <ForwardHyperGraph.h>`

Public Member Functions

- CycleEnumerator (const FHG &fhg, Collector &c)

  Constructor from a graph and a collector object.
- bool findAll ()

  Start the search for all cycles. Returns whether the search was finished (true) or aborted (false).

0.15.139.1 Detailed Description

template<typename FHG, typename Collector>
struct smtrat::CycleEnumerator< FHG, Collector >

This class encapsulates an algorithm for enumerating all cycles.
This class implements an adaption of the algorithm described in **[Johnson75]**. Unlike the algorithm described there, this version works on forward hypergraphs as implemented by ForwardHyperGraph.
Although the algorithm works recursively, this implementation can be used somewhat iteratively. The cycles are given to the caller using an collector functor. If this functor returns true, the search for more cycles is aborted immediately. Due to this pattern, there is no need to store the cycles within this class and thus the memory usage is minimized.

0.15.139.2 Constructor & Destructor Documentation

0.15.139.2.1 CycleEnumerator()

```
template<typename FHG , typename Collector >
smtrat::CycleEnumerator< FHG, Collector >::CycleEnumerator (
            const FHG & fhg,
            Collector & c )  [inline]
```

Constructor from a graph and a collector object.

0.15.139.3 Member Function Documentation

0.15.139.3.1 findAll()

```
template<typename FHG , typename Collector >
bool smtrat::CycleEnumerator< FHG, Collector >::findAll ( )
```

Start the search for all cycles. Returns whether the search was finished (true) or aborted (false).

0.15.140 benchmax::Database Class Reference

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

```
#include <Database.h>
```

Public Types

- using Index = std::size_t

  Dummy index type.

Public Member Functions

- Index addTool (const Tool *)

  Dummy.
- Index getToolID (const Tool *)

  Dummy.
- Index addFile (const fs::path &)

  Dummy.
- Index getFileID (const fs::path &)

Dummy.

- Index createBenchmark ()

  Dummy.

- Index addBenchmarkResult (Index, Index, Index, int, std::size_t)

  Dummy.

- void addBenchmarkAttribute (Index, const std::string &, const std::string &)

  Dummy.

0.15.140.1 Detailed Description

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

0.15.140.2 Member Typedef Documentation

0.15.140.2.1 Index `using benchmax::Database::Index = std::size_t`

Dummy index type.

0.15.140.3 Member Function Documentation

0.15.140.3.1 addBenchmarkAttribute()

```
void benchmax::Database::addBenchmarkAttribute (
    Index ,
    const std::string & ,
    const std::string & ) [inline]
```

Dummy.

0.15.140.3.2 addBenchmarkResult()

```
Index benchmax::Database::addBenchmarkResult (
    Index ,
    Index ,
    Index ,
    int ,
    std::size_t ) [inline]
```

Dummy.

0.15.140.3.3 addFile()

```
Index benchmax::Database::addFile (
    const fs::path & ) [inline]
```

Dummy.

0.15.140.3.4 addTool()

```
Index benchmax::Database::addTool (
    const Tool * ) [inline]
```

Dummy.

0.15.140.3.5 createBenchmark()

```
Index benchmax::Database::createBenchmark ( ) [inline]
```

Dummy.

0.15.140.3.6 getFileID()

```
Index benchmax::Database::getFileID (
    const fs::path & ) [inline]
```

Dummy.

0.15.140.3.7 getToolID()

```
Index benchmax::Database::getToolID (
    const Tool * ) [inline]
```

Dummy.

0.15.141 benchmax::DBAL Class Reference

```
#include <Database_mysqlconnector.h>
```

Public Types

- using Index = std::size_t

  Indec type.
- using Statement = std::unique_ptr< sql::PreparedStatement >

  Statement type.
- using Results = std::unique_ptr< sql::ResultSet >

  Results type.
- typedef mysqlpp::sql_bigint_unsigned Index
- typedef mysqlpp::Query Statement
- typedef mysqlpp::StoreQueryResult Results

Public Member Functions

- bool connect (const std::string &db, const std::string &host, const std::string &user, const std::string &password)

  Connect to the given database.
- Statement prepare (const std::string &query)

  Prepare a statement.
- Results execute (Statement &stmt)

  Execute a statement without arguments.
- Results execute (Statement &stmt, const std::string &a1)

  Execute a statement with a single string.
- Results execute (Statement &stmt, const std::string &a1, std::size_t a2)

  Execute a statement with a string and a number.
- Results execute (Statement &stmt, const std::string &a1, const std::string &a2, const Index &a3)

  Execute a statement with two strings and an index.
- Results execute (Statement &stmt, int a1, std::size_t a2, std::size_t a3, const Index &a4, const Index &a5, const Index &a6)

  Execute a statement with an int, two numbers and three indices.
- template<typename... Args>
  Index insert (const std::string &query, const Args &... args)

  Insert some data and return the new index.
- template<typename... Args>
  Results select (const std::string &query, const Args &... args)

  Select some data.
- std::size_t size (const Results &res)

  Return the size of the results.
- Index getIndex (const Results &res, const std::string &name)

  Get the index of some name within the results.

- bool connect (const std::string &db, const std::string &host, const std::string &user, const std::string &password)
- Statement prepare (const std::string &query)
- template<typename... Args>
  void execute (Statement &stmt, const Args &... args)
- template<typename... Args>
  Index insert (const std::string &query, const Args &... args)
- template<typename... Args>
  Results select (const std::string &query, const Args &... args)
- std::size_t size (const Results &res)
- Index getIndex (const Results &res, const std::string &name)

0.15.141.1 Member Typedef Documentation

0.15.141.1.1 Index [1/2]

```
using benchmax::DBAL::Index = std::size_t
```

Indec type.

0.15.141.1.2 Index [2/2]

```
typedef mysqlpp::sql_bigint_unsigned benchmax::DBAL::Index
```

0.15.141.1.3 Results [1/2]

```
using benchmax::DBAL::Results = std::unique_ptr<sql::ResultSet>
```

Results type.

0.15.141.1.4 Results [2/2]

```
typedef mysqlpp::StoreQueryResult benchmax::DBAL::Results
```

0.15.141.1.5 Statement [1/2]

```
using benchmax::DBAL::Statement = std::unique_ptr<sql::Prepared↩
Statement>
```

Statement type.

0.15.141.1.6 Statement [2/2]

```
typedef mysqlpp::Query benchmax::DBAL::Statement
```

0.15.141.2 Member Function Documentation

0.15.141.2.1 connect() [1/2]

```
bool benchmax::DBAL::connect (
    const std::string & db,
    const std::string & host,
    const std::string & user,
    const std::string & password ) [inline]
```

Connect to the given database.

0.15.141.2.2 connect() [2/2]

```
bool benchmax::DBAL::connect (
    const std::string & db,
    const std::string & host,
    const std::string & user,
    const std::string & password ) [inline]
```

0.15.141.2.3 execute() [1/6]

```
Results benchmax::DBAL::execute (
    Statement & stmt ) [inline]
```

Execute a statement without arguments.

0.15.141.2.4 execute() [2/6]

```
template<typename... Args>
void benchmax::DBAL::execute (
    Statement & stmt,
    const Args &... args ) [inline]
```

0.15.141.2.5 execute() [3/6]

```
Results benchmax::DBAL::execute (
    Statement & stmt,
    const std::string & a1 ) [inline]
```

Execute a statement with a single string.

0.15.141.2.6 execute() [4/6]

```
Results benchmax::DBAL::execute (
    Statement & stmt,
    const std::string & a1,
    const std::string & a2,
    const Index & a3 ) [inline]
```

Execute a statement with two strings and an index.

0.15.141.2.7 execute() [5/6]

```
Results benchmax::DBAL::execute (
    Statement & stmt,
    const std::string & a1,
    std::size_t a2 ) [inline]
```

Execute a statement with a string and a number.

0.15.141.2.8 execute() [6/6]

```
Results benchmax::DBAL::execute (
    Statement & stmt,
    int a1,
    std::size_t a2,
    std::size_t a3,
    const Index & a4,
    const Index & a5,
    const Index & a6 ) [inline]
```

Execute a statement with an int, two numbers and three indices.

0.15.141.2.9 getIndex() [1/2]

```
Index benchmax::DBAL::getIndex (
    const Results & res,
    const std::string & name ) [inline]
```

Get the index of some name within the results.

0.15.141.2.10 getIndex() [2/2]

```
Index benchmax::DBAL::getIndex (
    const Results & res,
    const std::string & name ) [inline]
```

0.15.141.2.11 insert() [1/2] template<typename... Args>

```
Index benchmax::DBAL::insert (
            const std::string & query,
            const Args &... args ) [inline]
```

Insert some data and return the new index.

0.15.141.2.12 insert() [2/2] template<typename... Args>

```
Index benchmax::DBAL::insert (
            const std::string & query,
            const Args &... args ) [inline]
```

0.15.141.2.13 prepare() [1/2]

```
Statement benchmax::DBAL::prepare (
            const std::string & query ) [inline]
```

Prepare a statement.

0.15.141.2.14 prepare() [2/2]

```
Statement benchmax::DBAL::prepare (
            const std::string & query ) [inline]
```

0.15.141.2.15 select() [1/2] template<typename... Args>

```
Results benchmax::DBAL::select (
            const std::string & query,
            const Args &... args ) [inline]
```

Select some data.

0.15.141.2.16 select() [2/2] template<typename... Args>

```
Results benchmax::DBAL::select (
            const std::string & query,
            const Args &... args ) [inline]
```

0.15.141.2.17 size() [1/2]

```
std::size_t benchmax::DBAL::size (
            const Results & res ) [inline]
```

Return the size of the results.

0.15.141.2.18 size() [2/2]

```
std::size_t benchmax::DBAL::size (
            const Results & res ) [inline]
```

0.15.142 smtrat::parser::DecimalParser Struct Reference

Parses decimals: `numeral.0*numeral`

`#include <Lexicon.h>`

0.15.142.1 Detailed Description

Parses decimals: `numeral.0*numeral`

0.15.143 Minisat::DeepEqual< K > Struct Template Reference

`#include <Map.h>`

Public Member Functions

- bool operator() (const K ∗k1, const K ∗k2) const

0.15.143.1 Member Function Documentation

0.15.143.1.1 operator()()

```
template<class K >
bool Minisat::DeepEqual< K >::operator() (
            const K * k1,
            const K * k2 ) const  [inline]
```

0.15.144 Minisat::DeepHash< K > Struct Template Reference

```
#include <Map.h>
```

Public Member Functions

- uint32_t operator() (const K ∗k) const

0.15.144.1 Member Function Documentation

0.15.144.1.1 operator()()

```
template<class K >
uint32_t Minisat::DeepHash< K >::operator() (
            const K * k ) const  [inline]
```

0.15.145 smtrat::mcsat::fm::DefaultComparator Struct Reference

Does not order anything.

```
#include <ConflictGenerator.h>
```

Public Member Functions

- bool operator() (const Bound &, const Bound &) const

Data Fields

- bool symmetric = false

0.15.145.1 Detailed Description

Does not order anything.

0.15.145.2 Member Function Documentation

0.15.145.2.1 operator()()

```
bool smtrat::mcsat::fm::DefaultComparator::operator() (
            const Bound & ,
            const Bound &  ) const  [inline]
```

0.15.145.3 Field Documentation

0.15.145.3.1 symmetric

```
bool smtrat::mcsat::fm::DefaultComparator::symmetric = false
```

0.15.146 smtrat::mcsat::fm::DefaultSettings Struct Reference

`#include <Explanation.h>`

Static Public Attributes

- static constexpr bool use_all_constraints = false

0.15.146.1 Field Documentation

0.15.146.1.1 use_all_constraints

`constexpr bool smtrat::mcsat::fm::DefaultSettings::use_all_constraints = false [static], [constexpr]`

0.15.147 smtrat::mcsat::smtaf::DefaultSettings Struct Reference

`#include <AssignmentFinder.h>`

Static Public Attributes

- static constexpr unsigned int assignAllVariables = true
- static constexpr bool advance_level_by_level = false

  If set to true, a conflict on the lowest possible level is returned.

0.15.147.1 Field Documentation

0.15.147.1.1 advance_level_by_level

`constexpr bool smtrat::mcsat::smtaf::DefaultSettings::advance_level_by_level = false [static], [constexpr]`

If set to true, a conflict on the lowest possible level is returned.
Got more or less irrelevant as unassigned variables are not ordered anymore.

0.15.147.1.2 assignAllVariables

`constexpr unsigned int smtrat::mcsat::smtaf::DefaultSettings::assignAllVariables = true [static], [constexpr]`

0.15.148 smtrat::mcsat::vs::DefaultSettings Struct Reference

`#include <ExplanationGenerator.h>`

Static Public Attributes

- static const bool reduceConflictConstraints = true
- static const bool clause_chain_with_equivalences = false

0.15.148.1 Field Documentation

0.15.148.1.1 clause_chain_with_equivalences

`const bool smtrat::mcsat::vs::DefaultSettings::clause_chain_with_equivalences = false [static]`

0.15.148.1.2 reduceConflictConstraints

`const bool smtrat::mcsat::vs::DefaultSettings::reduceConflictConstraints = true [static]`

0.15.149 smtrat::cad::projection_compare::degree Struct Reference

`#include <ProjectionComparator.h>`

0.15.150 smtrat::mcsat::onecellcad::recursive::DegreeAscending Struct Reference

`#include <Explanation.h>`

Static Public Attributes

- static constexpr int heuristic = 1

0.15.150.1 Field Documentation

0.15.150.1.1 heuristic `constexpr int smtrat::mcsat::onecellcad::recursive::DegreeAscending↩ ::heuristic = 1 [static], [constexpr]`

0.15.151 smtrat::analyzer::DegreeCollector Struct Reference

`#include <utils.h>`

Public Member Functions

- void operator() (const Poly &p)
- void operator() (const carl::UnivariatePolynomial< Poly > &p)
- void operator() (const ConstraintT &c)

Data Fields

- std::size_t degree_max = 0
- std::size_t degree_sum = 0
- std::size_t tdegree_max = 0
- std::size_t tdegree_sum = 0

0.15.151.1 Member Function Documentation

0.15.151.1.1 operator()() [1/3] `void smtrat::analyzer::DegreeCollector::operator() ( const carl::UnivariatePolynomial< Poly > & p ) [inline]`

0.15.151.1.2 operator()() [2/3] `void smtrat::analyzer::DegreeCollector::operator() ( const ConstraintT & c ) [inline]`

0.15.151.1.3 operator()() [3/3] `void smtrat::analyzer::DegreeCollector::operator() ( const Poly & p ) [inline]`

0.15.151.2 Field Documentation

0.15.151.2.1 degree_max `std::size_t smtrat::analyzer::DegreeCollector::degree_max = 0`

0.15.151.2.2 degree_sum `std::size_t smtrat::analyzer::DegreeCollector::degree_sum = 0`

0.15.151.2.3 tdegree_max `std::size_t smtrat::analyzer::DegreeCollector::tdegree_max = 0`

0.15.151.2.4 tdegree_sum `std::size_t smtrat::analyzer::DegreeCollector::tdegree_sum = 0`

0.15.152 smtrat::mcsat::onecellcad::recursive::DegreeDescending Struct Reference

`#include <Explanation.h>`

Static Public Attributes

- static constexpr int heuristic = 2

0.15.152.1 Field Documentation

0.15.152.1.1 heuristic `constexpr int smtrat::mcsat::onecellcad::recursive::DegreeDescending↩ ::heuristic = 2 [static], [constexpr]`

0.15.153 smtrat::cadcells::datastructures::DelineatedDerivation< Properties > Class Template Reference

A DelineatedDerivation is a BaseDerivation with a Delineation and an underlying SampledDerivation.
`#include <derivation.h>`

Public Member Functions

- DelineatedDerivation (BaseDerivationRef< Properties > base)
- DelineatedDerivation (BaseDerivationRef< Properties > base, const Delineation &delineation)
- BaseDerivationRef< Properties > & base ()
- const BaseDerivationRef< Properties > & base () const
- Delineation & delin ()
- const Assignment & underlying_sample () const
- DerivationRef< Properties > & underlying ()
- const DerivationRef< Properties > & underlying () const
- PolyPool & polys ()
- Projections & proj ()
- carl::Variable main_var () const
- size_t level () const
- template<typename P >
  void insert (P property)
- template<typename P >
  bool contains (const P &property) const
- template<typename P >
  const PropertiesTSet< P > & properties () const
- void merge_with (const DelineatedDerivation< Properties > &other)

0.15.153.1 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

0.15.153.2 Constructor & Destructor Documentation

0.15.153.2.1 DelineatedDerivation() [1/2]

```
template<typename Properties >
smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::DelineatedDerivation (
            BaseDerivationRef< Properties > base )  [inline]
```

0.15.153.2.2 DelineatedDerivation() [2/2]

```
template<typename Properties >
smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::DelineatedDerivation (
            BaseDerivationRef< Properties > base,
            const Delineation & delineation )  [inline]
```

0.15.153.3 Member Function Documentation

0.15.153.3.1 base() [1/2]

```
template<typename Properties >
BaseDerivationRef<Properties>& smtrat::cadcells::datastructures::DelineatedDerivation< Properties
>::base ( )  [inline]
```

0.15.153.3.2 base() [2/2]

```
template<typename Properties >
const BaseDerivationRef<Properties>& smtrat::cadcells::datastructures::DelineatedDerivation<
Properties >::base ( ) const  [inline]
```

0.15.153.3.3 contains()

```
template<typename Properties >
template<typename P >
bool smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::contains (
            const P & property ) const  [inline]
```

0.15.153.3.4 delin()

```
template<typename Properties >
Delineation& smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::delin ( )
[inline]
```

0.15.153.3.5 insert()

```
template<typename Properties >
template<typename P >
void smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::insert (
            P property )  [inline]
```

0.15.153.3.6 level()

```
template<typename Properties >
size_t smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::level ( ) const
[inline]
```

0.15.153.3.7 main_var() template<typename Properties >

carl::Variable smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::main_var ( ) const [inline]

0.15.153.3.8 merge_with() template<typename Properties >

void smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::merge_with (const DelineatedDerivation< Properties > & *other*) [inline]

0.15.153.3.9 polys() template<typename Properties >

PolyPool& smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::polys ( ) [inline]

0.15.153.3.10 proj() template<typename Properties >

Projections& smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::proj ( ) [inline]

0.15.153.3.11 properties() template<typename Properties >

template<typename P >
const PropertiesTSet<P>& smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::properties ( ) const [inline]

0.15.153.3.12 underlying() [1/2] template<typename Properties >

DerivationRef<Properties>& smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::underlying ( ) [inline]

0.15.153.3.13 underlying() [2/2] template<typename Properties >

const DerivationRef<Properties>& smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::underlying ( ) const [inline]

0.15.153.3.14 underlying_sample() template<typename Properties >

const Assignment& smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::underlying↩_sample ( ) const [inline]

0.15.154 smtrat::cadcells::datastructures::Delineation Class Reference

Represents the delineation of a set of polynomials (at a sample), that is.

#include <delineation.h>

Public Member Functions

- Delineation ()
- const auto & roots () const

  Returns the underlying root map, which is a set of real algebraic numbers to indexed root expressions.
- const auto & nullified () const

  The set of nullified polynomials.
- const auto & nonzero () const

  The set of polynomials without roots.
- bool empty () const

- auto delineate_cell (const RAN &sample) const

  Computes the bounds of the interval around the given sample w.r.t.
- void add_root (RAN root, TaggedIndexedRoot tagged_root)
- void add_poly_nonzero (PolyRef poly)
- void add_poly_nullified (PolyRef poly)

Friends

- class DelineationInterval

0.15.154.1 Detailed Description

Represents the delineation of a set of polynomials (at a sample), that is.

- the ordering of all roots,
- the set of nullified polynomials,
- the set of polynomials without a root.

0.15.154.2 Constructor & Destructor Documentation

0.15.154.2.1 Delineation() `smtrat::cadcells::datastructures::Delineation::Delineation ( ) [inline]`

0.15.154.3 Member Function Documentation

0.15.154.3.1 add_poly_nonzero() `void smtrat::cadcells::datastructures::Delineation::add_poly_nonzero ( PolyRef poly ) [inline]`

0.15.154.3.2 add_poly_nullified() `void smtrat::cadcells::datastructures::Delineation::add_poly_nullified ( PolyRef poly ) [inline]`

0.15.154.3.3 add_root() `void smtrat::cadcells::datastructures::Delineation::add_root ( RAN root, TaggedIndexedRoot tagged_root ) [inline]`

0.15.154.3.4 delineate_cell() `auto smtrat::cadcells::datastructures::Delineation::delineate_cell ( const RAN & sample ) const [inline]`

Computes the bounds of the interval around the given sample w.r.t.
this delineation.

0.15.154.3.5 empty() `bool smtrat::cadcells::datastructures::Delineation::empty ( ) const [inline]`

0.15.154.3.6 nonzero() `const auto& smtrat::cadcells::datastructures::Delineation::nonzero ( ) const [inline]`

The set of polynomials without roots.

0.15.154.3.7 nullified() `const auto& smtrat::cadcells::datastructures::Delineation::nullified ( ) const [inline]`

The set of nullified polynomials.

0.15.154.3.8 roots() `const auto& smtrat::cadcells::datastructures::Delineation::roots ( ) const [inline]`

Returns the underlying root map, which is a set of real algebraic numbers to indexed root expressions.

0.15.154.4 Friends And Related Function Documentation

0.15.154.4.1 DelineationInterval `friend class DelineationInterval [friend]`

0.15.155 smtrat::cadcells::datastructures::DelineationInterval Class Reference

An interval of a delineation.

`#include <delineation.h>`

Public Member Functions

- bool is_section () const
- bool is_sector () const
- const auto & lower () const
- bool lower_unbounded () const
- bool lower_strict () const
- const auto & upper () const
- bool upper_unbounded () const
- bool upper_strict () const
- bool contains (const RAN &sample) const

Friends

- class Delineation

0.15.155.1 Detailed Description

An interval of a delineation.

0.15.155.2 Member Function Documentation

0.15.155.2.1 contains() `bool smtrat::cadcells::datastructures::DelineationInterval::contains ( const RAN & sample ) const [inline]`

0.15.155.2.2 is_section() `bool smtrat::cadcells::datastructures::DelineationInterval::is_section ( ) const [inline]`

0.15.155.2.3 is_sector() `bool smtrat::cadcells::datastructures::DelineationInterval::is_sector ( ) const [inline]`

0.15.155.2.4 lower() `const auto& smtrat::cadcells::datastructures::DelineationInterval::lower ( ) const [inline]`

0.15.155.2.5 lower_strict() `bool smtrat::cadcells::datastructures::DelineationInterval::lower_strict ( ) const [inline]`

0.15.155.2.6 lower_unbounded() `bool smtrat::cadcells::datastructures::DelineationInterval::lower_unbounded ( ) const [inline]`

0.15.155.2.7 upper() `const auto& smtrat::cadcells::datastructures::DelineationInterval::upper ( ) const [inline]`

0.15.155.2.8 upper_strict() `bool smtrat::cadcells::datastructures::DelineationInterval::upper_strict ( ) const [inline]`

0.15.155.2.9 upper_unbounded() `bool smtrat::cadcells::datastructures::DelineationInterval::upper_unbounded ( ) const [inline]`

0.15.155.3 Friends And Related Function Documentation

0.15.155.3.1 Delineation `friend class Delineation [friend]`

0.15.156 smtrat::mcsat::icp::Dependencies Class Reference

`#include <Dependencies.h>`

Public Member Functions

- template<typename Contractor >
  void add (const Contractor &c)
- void add (carl::Variable v, const FormulaT &c, const std::vector< carl::Variable > &used)
- std::vector< FormulaT > get (carl::Variable v, bool negate=true) const

0.15.156.1 Member Function Documentation

0.15.156.1.1 add() [1/2]

```
void smtrat::mcsat::icp::Dependencies::add (
            carl::Variable v,
            const FormulaT & c,
            const std::vector< carl::Variable > & used ) [inline]
```

0.15.156.1.2 add() [2/2]

```
template<typename Contractor >
void smtrat::mcsat::icp::Dependencies::add (
            const Contractor & c ) [inline]
```

0.15.156.1.3 get()

```
std::vector<FormulaT> smtrat::mcsat::icp::Dependencies::get (
    carl::Variable v,
    bool negate = true ) const [inline]
```

0.15.157 smtrat::cadcells::datastructures::DerivationRef< Properties > Class Template Reference

A reference to a derivation, which is either.

`#include <derivation.h>`

Public Member Functions

- DerivationRef (const BaseDerivationRef< Properties > &data)
- DerivationRef (const DelineatedDerivationRef< Properties > &data)
- DerivationRef (const SampledDerivationRef< Properties > &data)
- bool is_null () const
- bool is_sampled () const
- auto & base_ref ()
- auto & delineated_ref ()
- auto & sampled_ref ()
- const auto & base_ref () const
- const auto & delineated_ref () const
- const auto & sampled_ref () const
- auto & base ()
- auto & delineated ()
- auto & sampled ()
- const auto & base () const
- const auto & delineated () const
- const auto & sampled () const

Friends

- template<typename P >
  bool operator== (const DerivationRef< P > &lhs, const DerivationRef< P > &rhs)
- template<typename P >
  bool operator< (const DerivationRef< P > &lhs, const DerivationRef< P > &rhs)

0.15.157.1 Detailed Description

template<typename Properties>
class smtrat::cadcells::datastructures::DerivationRef< Properties >

A reference to a derivation, which is either.

- a BaseDerivation (a set of properties for each level)
- a DelineatedDerivation (a BaseDerivation with an underlying SampledDerivation on which the properties can be delineated)
- a SampledDerivation (a DelineatedDerivation with an underlying SampledDerivation plus a sample on the current level such that a delineated interval can be computed)

0.15.157.1.1 Memory management Memory management is based on std::shared_ptr. A SampledDerivation holds a shared pointer to a DelineatedDerivation, which in turn holds a shared pointer to BaseDerivation. The BaseDerivation holds a DerivationRef to the underlying derivation. Note that not all combinations are legal; i.e. the underlying derivation of a delineated derivation must be a sampled derivation.

0.15.157.2 Constructor & Destructor Documentation

0.15.157.2.1 DerivationRef() [1/3]

```
template<typename Properties >
smtrat::cadcells::datastructures::DerivationRef< Properties >::DerivationRef (
            const BaseDerivationRef< Properties > & data ) [inline]
```

0.15.157.2.2 DerivationRef() [2/3]

```
template<typename Properties >
smtrat::cadcells::datastructures::DerivationRef< Properties >::DerivationRef (
            const DelineatedDerivationRef< Properties > & data ) [inline]
```

0.15.157.2.3 DerivationRef() [3/3]

```
template<typename Properties >
smtrat::cadcells::datastructures::DerivationRef< Properties >::DerivationRef (
            const SampledDerivationRef< Properties > & data ) [inline]
```

0.15.157.3 Member Function Documentation

0.15.157.3.1 base() [1/2]

```
template<typename Properties >
auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::base ( ) [inline]
```

0.15.157.3.2 base() [2/2]

```
template<typename Properties >
const auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::base ( ) const
[inline]
```

0.15.157.3.3 base_ref() [1/2]

```
template<typename Properties >
auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::base_ref ( ) [inline]
```

0.15.157.3.4 base_ref() [2/2]

```
template<typename Properties >
const auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::base_ref ( ) const
[inline]
```

0.15.157.3.5 delineated() [1/2]

```
template<typename Properties >
auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::delineated ( ) [inline]
```

0.15.157.3.6 delineated() [2/2]

```
template<typename Properties >
const auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::delineated ( )
const [inline]
```

0.15.157.3.7 delineated_ref() [1/2]

```
template<typename Properties >
auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::delineated_ref ( ) [inline]
```

0.15.157.3.8 delineated_ref() [2/2]

```
template<typename Properties >
const auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::delineated_ref ( )
const [inline]
```

0.15.157.3.9 is_null() template<typename Properties >

bool smtrat::cadcells::datastructures::DerivationRef< Properties >::is_null ( ) const [inline]

0.15.157.3.10 is_sampled() template<typename Properties >

bool smtrat::cadcells::datastructures::DerivationRef< Properties >::is_sampled ( ) const [inline]

0.15.157.3.11 sampled() [1/2] template<typename Properties >

auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::sampled ( ) [inline]

0.15.157.3.12 sampled() [2/2] template<typename Properties >

const auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::sampled ( ) const [inline]

0.15.157.3.13 sampled_ref() [1/2] template<typename Properties >

auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::sampled_ref ( ) [inline]

0.15.157.3.14 sampled_ref() [2/2] template<typename Properties >

const auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::sampled_ref ( ) const [inline]

0.15.157.4 Friends And Related Function Documentation

0.15.157.4.1 operator< template<typename Properties >

```
template<typename P >
bool operator< (
            const DerivationRef< P > & lhs,
            const DerivationRef< P > & rhs ) [friend]
```

0.15.157.4.2 operator== template<typename Properties >

```
template<typename P >
bool operator== (
            const DerivationRef< P > & lhs,
            const DerivationRef< P > & rhs ) [friend]
```

0.15.158 smtrat::mcsat::onecellcad::recursive::DontCoverNullification Struct Reference

#include <Explanation.h>

Static Public Attributes

- static constexpr bool cover_nullification = false

0.15.158.1 Field Documentation

0.15.158.1.1 cover_nullification constexpr bool smtrat::mcsat::onecellcad::recursive::DontCover↩Nullification::cover_nullification = false [static], [constexpr]

0.15.159 smtrat::cad::debug::DotSubgraph Struct Reference

`#include <DotHelper.h>`

Public Member Functions

- DotSubgraph (const std::string &n)
- void add (const std::string &n)

Data Fields

- std::string name
- std::vector< std::string > nodes

Friends

- std::ostream & operator<< (std::ostream &os, const DotSubgraph &dsg)

0.15.159.1 Constructor & Destructor Documentation

0.15.159.1.1 DotSubgraph()

```
smtrat::cad::debug::DotSubgraph::DotSubgraph (
const std::string & n ) [inline]
```

0.15.159.2 Member Function Documentation

0.15.159.2.1 add()

```
void smtrat::cad::debug::DotSubgraph::add (
const std::string & n ) [inline]
```

0.15.159.3 Friends And Related Function Documentation

0.15.159.3.1 operator<<

```
std::ostream& operator<< (
std::ostream & os,
const DotSubgraph & dsg ) [friend]
```

0.15.159.4 Field Documentation

0.15.159.4.1 name

```
std::string smtrat::cad::debug::DotSubgraph::name
```

0.15.159.4.2 nodes

```
std::vector<std::string> smtrat::cad::debug::DotSubgraph::nodes
```

0.15.160 Minisat::DoubleOption Class Reference

`#include <Options.h>`

Public Member Functions

- DoubleOption (const char *c, const char *n, const char *d, double def=double(), DoubleRange r=DoubleRange(-HUGE_VAL, false, HUGE_VAL, false))
- operator double (void) const
- DoubleOption & operator= (double x)
- virtual void help (bool verbose=false)

Static Protected Member Functions

- static vec< Option * > & getOptionList ()
- static const char *& getUsageString ()
- static const char *& getHelpPrefixString ()

Protected Attributes

- DoubleRange range
- double value
- const char * name
- const char * description
- const char * category
- const char * type_name

0.15.160.1 Constructor & Destructor Documentation

0.15.160.1.1 DoubleOption()

```
Minisat::DoubleOption::DoubleOption (
    const char * c,
    const char * n,
    const char * d,
    double def = double(),
    DoubleRange r = DoubleRange( -HUGE_VAL, false, HUGE_VAL, false ) )  [inline]
```

0.15.160.2 Member Function Documentation

0.15.160.2.1 getHelpPrefixString()

```
static const char*& Minisat::Option::getHelpPrefixString ( )
[inline], [static], [protected], [inherited]
```

0.15.160.2.2 getOptionList()

```
static vec<Option*>& Minisat::Option::getOptionList ( )  [inline],
[static], [protected], [inherited]
```

0.15.160.2.3 getUsageString()

```
static const char*& Minisat::Option::getUsageString ( )  [inline],
[static], [protected], [inherited]
```

0.15.160.2.4 help()

```
virtual void Minisat::DoubleOption::help (
    bool verbose = false )  [inline], [virtual]
```

Implements Minisat::Option.

0.15.160.2.5 operator double()

```
Minisat::DoubleOption::operator double (
    void  ) const  [inline]
```

0.15.160.2.6 operator=() `DoubleOption& Minisat::DoubleOption::operator= ( double x ) [inline]`

0.15.160.3 Field Documentation

0.15.160.3.1 category `const char* Minisat::Option::category [protected], [inherited]`

0.15.160.3.2 description `const char* Minisat::Option::description [protected], [inherited]`

0.15.160.3.3 name `const char* Minisat::Option::name [protected], [inherited]`

0.15.160.3.4 range `DoubleRange Minisat::DoubleOption::range [protected]`

0.15.160.3.5 type_name `const char* Minisat::Option::type_name [protected], [inherited]`

0.15.160.3.6 value `double Minisat::DoubleOption::value [protected]`

0.15.161 Minisat::DoubleRange Struct Reference

`#include <Options.h>`

Public Member Functions

- DoubleRange (double b, bool binc, double e, bool einc)

Data Fields

- double begin
- double end
- bool begin_inclusive
- bool end_inclusive

0.15.161.1 Constructor & Destructor Documentation

0.15.161.1.1 DoubleRange() `Minisat::DoubleRange::DoubleRange ( double b, bool binc, double e, bool einc ) [inline]`

0.15.161.2 Field Documentation

0.15.161.2.1 begin `double Minisat::DoubleRange::begin`

0.15.161.2.2 begin_inclusive `bool Minisat::DoubleRange::begin_inclusive`

0.15.161.2.3 end `double Minisat::DoubleRange::end`

0.15.161.2.4 end_inclusive `bool Minisat::DoubleRange::end_inclusive`

0.15.162 smtrat::expression::simplifier::DuplicateSimplifier Struct Reference

`#include <DuplicateSimplifier.h>`

Public Member Functions

- const ExpressionContent * simplify (const NaryExpression &expr) const
- const ExpressionContent * operator() (const carl::Variable &expr) const
- const ExpressionContent * operator() (const ITEExpression &expr) const
- const ExpressionContent * operator() (const QuantifierExpression &expr) const
- const ExpressionContent * operator() (const UnaryExpression &expr) const
- const ExpressionContent * operator() (const BinaryExpression &expr) const
- const ExpressionContent * operator() (const NaryExpression &expr) const
- const ExpressionContent * operator() (const ExpressionContent *_ec) const

Protected Member Functions

- virtual const ExpressionContent * simplify (const carl::Variable &) const
- virtual const ExpressionContent * simplify (const ITEExpression &) const
- virtual const ExpressionContent * simplify (const QuantifierExpression &) const
- virtual const ExpressionContent * simplify (const UnaryExpression &) const
- virtual const ExpressionContent * simplify (const BinaryExpression &) const

0.15.162.1 Member Function Documentation

0.15.162.1.1 operator()() [1/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const BinaryExpression & expr ) const  [inline], [inherited]
```

0.15.162.1.2 operator()() [2/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const carl::Variable & expr ) const  [inline], [inherited]
```

0.15.162.1.3 operator()() [3/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const ExpressionContent * _ec ) const  [inline], [inherited]
```

0.15.162.1.4 operator()() [4/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const ITEExpression & expr ) const  [inline], [inherited]
```

0.15.162.1.5 operator()() [5/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const NaryExpression & expr ) const  [inline], [inherited]
```

0.15.162.1.6 operator()() [6/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const QuantifierExpression & expr ) const  [inline], [inherited]
```

0.15.162.1.7 operator()() [7/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const UnaryExpression & expr ) const  [inline], [inherited]
```

0.15.162.1.8 simplify() [1/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const BinaryExpression &  ) const  [inline], [protected], [virtual], [inherited]
```

0.15.162.1.9 simplify() [2/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const carl::Variable &  ) const  [inline], [protected], [virtual], [inherited]
```

0.15.162.1.10 simplify() [3/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const ITEExpression &  ) const  [inline], [protected], [virtual], [inherited]
```

0.15.162.1.11 simplify() [4/6]

```
const ExpressionContent* smtrat::expression::simplifier::DuplicateSimplifier::simplify (
            const NaryExpression & expr ) const  [inline], [virtual]
```

Reimplemented from smtrat::expression::simplifier::BaseSimplifier.

0.15.162.1.12 simplify() [5/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const QuantifierExpression &  ) const  [inline], [protected], [virtual], [inherited]
```

0.15.162.1.13 simplify() [6/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const UnaryExpression &  ) const  [inline], [protected], [virtual], [inherited]
```

Reimplemented in smtrat::expression::simplifier::NegationSimplifier.

0.15.163 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

0.15.163.1 Member Typedef Documentation

0.15.163.1.1 value_type

```
template<typename T , typename Compare = std::less<T>>
using smtrat::DynamicPriorityQueue< T, Compare >::value_type = typename Sequence::value_type
```

0.15.163.2 Constructor & Destructor Documentation

0.15.163.2.1 DynamicPriorityQueue() [1/2]

```
template<typename T , typename Compare = std::less<↩
T>>
smtrat::DynamicPriorityQueue< T, Compare >::DynamicPriorityQueue (
            const Compare & _comp = Compare() )  [inline], [explicit]
```

0.15.163.2.2 DynamicPriorityQueue() [2/2]

```
template<typename T , typename Compare = std::less<↩
T>>
smtrat::DynamicPriorityQueue< T, Compare >::DynamicPriorityQueue (
            Sequence && _seq,
            const Compare & _comp = Compare() )  [inline], [explicit]
```

0.15.163.3 Member Function Documentation

0.15.163.3.1 emplace()

```
template<typename T , typename Compare = std::less<T>>
template<typename...  Args>
void smtrat::DynamicPriorityQueue< T, Compare >::emplace (
            Args &&...  args )  [inline]
```

0.15.163.3.2 empty()

```
template<typename T , typename Compare = std::less<T>>
bool smtrat::DynamicPriorityQueue< T, Compare >::empty ( ) const  [inline]
```

0.15.163.3.3 fix()

```
template<typename T , typename Compare = std::less<T>>
void smtrat::DynamicPriorityQueue< T, Compare >::fix ( ) [inline]
```

0.15.163.3.4 pop()

```
template<typename T , typename Compare = std::less<T>>
void smtrat::DynamicPriorityQueue< T, Compare >::pop ( ) [inline]
```

0.15.163.3.5 popTop()

```
template<typename T , typename Compare = std::less<T>>
T smtrat::DynamicPriorityQueue< T, Compare >::popTop ( ) [inline]
```

0.15.163.3.6 push() [1/2]

```
template<typename T , typename Compare = std::less<T>>
void smtrat::DynamicPriorityQueue< T, Compare >::push (
            const T & t ) [inline]
```

0.15.163.3.7 push() [2/2]

```
template<typename T , typename Compare = std::less<T>>
void smtrat::DynamicPriorityQueue< T, Compare >::push (
            T && t ) [inline]
```

0.15.163.3.8 size()

```
template<typename T , typename Compare = std::less<T>>
std::size_t smtrat::DynamicPriorityQueue< T, Compare >::size ( ) const [inline]
```

0.15.163.3.9 top()

```
template<typename T , typename Compare = std::less<T>>
const T& smtrat::DynamicPriorityQueue< T, Compare >::top ( ) const [inline]
```

0.15.163.4 Field Documentation

0.15.163.4.1 c

```
template<typename T , typename Compare = std::less<T>>
Sequence smtrat::DynamicPriorityQueue< T, Compare >::c [protected]
```

0.15.163.4.2 comp

```
template<typename T , typename Compare = std::less<T>>
Compare smtrat::DynamicPriorityQueue< T, Compare >::comp [protected]
```

0.15.164 smtrat::cad::ProjectionGlobalInformation::ECData Struct Reference

```
#include <ProjectionInformation.h>
```

Data Fields

- std::size_t level = 0
- carl::Bitset polynomials

0.15.164.1 Field Documentation

0.15.164.1.1 level

```
std::size_t smtrat::cad::ProjectionGlobalInformation::ECData::level = 0
```

0.15.164.1.2 polynomials `carl::Bitset smtrat::cad::ProjectionGlobalInformation::ECData::polynomials`

0.15.165 smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Edge Struct Reference

Internal type of an edge.
`#include <ForwardHyperGraph.h>`

Public Member Functions

- Edge (const EdgeProperty &ep)

  Constructor from a property.
- void addOut (Vertex v)

  Add v to the outgoing vertices of this edge.
- const EdgeProperty & operator∗ () const

  Access the property.
- EdgeProperty & operator∗ ()

  Access the property.
- const EdgeProperty ∗ operator-> () const

  Access the property.
- EdgeProperty ∗ operator-> ()

  Access the property.
- const std::vector< Vertex > & out () const

  Retrieve the list of outgoing vertices.

Data Fields

- friend ForwardHyperGraph

0.15.165.1 Detailed Description

template<typename VertexProperty, typename EdgeProperty, bool UniqueVertices = true>
struct smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Edge

Internal type of an edge.

0.15.165.2 Constructor & Destructor Documentation

0.15.165.2.1 Edge()

```
template<typename VertexProperty , typename EdgeProperty , bool Unique↩
Vertices = true>
smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Edge::Edge (
            const EdgeProperty & ep )  [inline]
```

Constructor from a property.

0.15.165.3 Member Function Documentation

0.15.165.3.1 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.

0.15.165.3.2 operator*() [1/2]

```
template<typename VertexProperty , typename EdgeProperty , bool UniqueVertices = true>
EdgeProperty& smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >↩
::Edge::operator* ( ) [inline]
```

Access the property.

0.15.165.3.3 operator*() [2/2]

```
template<typename VertexProperty , typename EdgeProperty , bool UniqueVertices = true>
const EdgeProperty& smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices
>::Edge::operator* ( ) const [inline]
```

Access the property.

0.15.165.3.4 operator->() [1/2]

```
template<typename VertexProperty , typename EdgeProperty , bool UniqueVertices = true>
EdgeProperty* smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >↩
::Edge::operator-> ( ) [inline]
```

Access the property.

0.15.165.3.5 operator->() [2/2]

```
template<typename VertexProperty , typename EdgeProperty , bool UniqueVertices = true>
const EdgeProperty* smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices
>::Edge::operator-> ( ) const [inline]
```

Access the property.

0.15.165.3.6 out()

```
template<typename VertexProperty , typename EdgeProperty , bool Unique↩
Vertices = true>
const std::vector<Vertex>& smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, Unique↩
Vertices >::Edge::out ( ) const [inline]
```

Retrieve the list of outgoing vertices.

0.15.165.4 Field Documentation

0.15.165.4.1 ForwardHyperGraph

```
template<typename VertexProperty , typename EdgeProperty , bool UniqueVertices = true>
friend smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Edge::↩
ForwardHyperGraph
```

0.15.166 smtrat::ElementWithOrigins< Element, Origin > Class Template Reference

```
#include <IntBlastModule.h>
```

Public Member Functions

- ElementWithOrigins (const Element &_element)
- ElementWithOrigins (const Element &_element, const Origin &_origin)
- const Element & element () const
- const std::set< Origin > & origins () const
- void addOrigin (const Origin &_origin)
- bool removeOrigin (const Origin &_origin)
- bool hasOrigins ()

0.15.166.1 Constructor & Destructor Documentation

0.15.166.1.1 ElementWithOrigins() [1/2]

```
template<typename Element , typename Origin >
smtrat::ElementWithOrigins< Element, Origin >::ElementWithOrigins (
            const Element & _element )  [inline]
```

0.15.166.1.2 ElementWithOrigins() [2/2]

```
template<typename Element , typename Origin >
smtrat::ElementWithOrigins< Element, Origin >::ElementWithOrigins (
            const Element & _element,
            const Origin & _origin )  [inline]
```

0.15.166.2 Member Function Documentation

0.15.166.2.1 addOrigin()

```
template<typename Element , typename Origin >
void smtrat::ElementWithOrigins< Element, Origin >::addOrigin (
            const Origin & _origin )  [inline]
```

0.15.166.2.2 element()

```
template<typename Element , typename Origin >
const Element& smtrat::ElementWithOrigins< Element, Origin >::element ( ) const  [inline]
```

0.15.166.2.3 hasOrigins()

```
template<typename Element , typename Origin >
bool smtrat::ElementWithOrigins< Element, Origin >::hasOrigins ( )  [inline]
```

0.15.166.2.4 origins()

```
template<typename Element , typename Origin >
const std::set<Origin>& smtrat::ElementWithOrigins< Element, Origin >::origins ( ) const
[inline]
```

0.15.166.2.5 removeOrigin()

```
template<typename Element , typename Origin >
bool smtrat::ElementWithOrigins< Element, Origin >::removeOrigin (
            const Origin & _origin )  [inline]
```

0.15.167 smtrat::EMModule< Settings > Class Template Reference

```
#include <EMModule.h>
```

Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- EMModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=NULL)
- ~EMModule ()
- Answer checkCore ()

  Checks the received formula for consistency.
- bool isPreprocessor () const
- bool appliedPreprocessing () const

- bool add (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- void remove (ModuleInput::const_iterator _subformula)

  Removes everything related to the given sub-formula of the received formula.
- Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)

  Checks the received formula for consistency.
- Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- std::pair< bool, FormulaT > getReceivedFormulaSimplified ()
- void updateModel () const

  Updates the current assignment into the model.
- bool inform (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void deinform (const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual void init ()

  Informs all backends about the so far encountered constraints, which have not yet been communicated.
- virtual void updateAllModels ()

  Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
- virtual std::list< std::vector< carl::Variable > > getModelEqualities () const

  Partition the variables from the current model into equivalence classes according to their assigned value.
- unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const
- virtual unsigned currentlySatisfied (const FormulaT &) const
- bool receivedVariable (carl::Variable::Arg _var) const
- Answer solverState () const
- std::size_t id () const
- void setId (std::size_t _id)

  Sets this modules unique ID to identify itself.
- thread_priority threadPriority () const
- void setThreadPriority (thread_priority _threadPriority)

  Sets the priority of this module to get a thread for running its check procedure.
- const ModuleInput * pReceivedFormula () const
- const ModuleInput & rReceivedFormula () const
- const ModuleInput * pPassedFormula () const
- const ModuleInput & rPassedFormula () const
- const Model & model () const
- const std::vector< Model > & allModels () const
- const std::vector< FormulaSetT > & infeasibleSubsets () const
- const std::vector< Module * > & usedBackends () const
- const carl::FastSet< FormulaT > & constraintsToInform () const
- const carl::FastSet< FormulaT > & informedConstraints () const
- void addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))

  Stores a lemma being a valid formula.
- bool hasLemmas ()

  Checks whether this module or any of its backends provides any lemmas.
- void clearLemmas ()

  Deletes all yet found lemmas.
- const std::vector< Lemma > & lemmas () const
- ModuleInput::const_iterator firstUncheckedReceivedSubformula () const
- ModuleInput::const_iterator firstSubformulaToPass () const

- void receivedFormulaChecked ()

  Notifies that the received formulas has been checked.
- const 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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.
- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)
- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

Copies the infeasible subsets of the passed formula.

- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

Stores the assignment of the current satisfiable result, if existent.

- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.167.1 Member Typedef Documentation

0.15.167.1.1 SettingsType `template<typename Settings >`

```
typedef Settings smtrat::EMModule< Settings >::SettingsType
```

0.15.167.2 Member Enumeration Documentation

0.15.167.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

0.15.167.3 Constructor & Destructor Documentation

0.15.167.3.1 EMModule() `template<typename Settings >`

```
smtrat::EMModule< Settings >::EMModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = NULL )
```

0.15.167.3.2 ~EMModule() `template<typename Settings >`

```
smtrat::EMModule< Settings >::~EMModule ( )
```

0.15.167.4 Member Function Documentation

0.15.167.4.1 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.

0.15.167.4.2 addConstraintToInform() `void smtrat::Module::addConstraintToInform (`

```
            const FormulaT & _constraint )  [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

0.15.167.4.3 addCore() `virtual bool smtrat::Module::addCore (`

```
            [[maybe_unused] ] 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.

0.15.167.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.167.4.5 addLemma()

```
void smtrat::Module::addLemma (
    const FormulaT & _lemma,
    const LemmaType & _lt = LemmaType::NORMAL,
    const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline],
[inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.167.4.6 addOrigin()

```
void smtrat::Module::addOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.167.4.7 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.167.4.8 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula )  [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.167.4.9 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula,
            const FormulaT & _origin )  [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.167.4.10 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.

0.15.167.4.11 allModels() `const std::vector<Model>& smtrat::Module::allModels ( ) const [inline], [inherited]`

Returns

All satisfying assignments, if existent.

0.15.167.4.12 anAnswerFound() `bool smtrat::Module::anAnswerFound ( ) const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.167.4.13 answerFound() `const smtrat::Conditionals& smtrat::Module::answerFound ( ) const [inline], [inherited]`

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.167.4.14 appliedPreprocessing() `bool smtrat::PModule::appliedPreprocessing ( ) const [inline], [inherited]`

Returns

true, if the current received formula has been simplified and the result of this simplification is stored in the passed formula.

0.15.167.4.15 backendsModel() `const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.167.4.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.167.4.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.167.4.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline],
[protected], [inherited]
```

0.15.167.4.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.167.4.20 check()

```
Answer smtrat::PModule::check (
    bool _final = false,
    bool _full = true,
    carl::Variable _objective = carl::Variable::NO_VARIABLE ) [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.167.4.21 checkCore()

`template<typename Settings >`
`Answer smtrat::EMModule< Settings >::checkCore ( ) [virtual]`

Checks the received formula for consistency.

Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.167.4.22 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.

0.15.167.4.23 checkModel()

`unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]`

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.167.4.24 cleanModel()

`void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.167.4.25 clearLemmas()

`void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

0.15.167.4.26 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

0.15.167.4.27 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

0.15.167.4.28 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.167.4.29 collectOrigins() [1/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

0.15.167.4.30 collectOrigins() [2/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inherited]`

Collects the formulas in the given formula, which are part of the received formula.
If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.167.4.31 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

0.15.167.4.32 constraintsToInform() `const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns

The constraints which the backends must be informed about.

0.15.167.4.33 currentlySatisfied() `virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.167.4.34 currentlySatisfiedByBackend()

```
unsigned smtrat::Module::currentlySatisfiedByBackend (
    const FormulaT & _formula ) const [inherited]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.167.4.35 deinform()

```
void smtrat::Module::deinform (
    const FormulaT & _constraint ) [inherited]
```

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.167.4.36 deinformCore()

```
virtual void smtrat::Module::deinformCore (
    [[maybe_unused] ] 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.

0.15.167.4.37 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

0.15.167.4.38 eraseSubformulaFromPassedFormula()

```
ModuleInput::iterator smtrat::Module::erase↩
SubformulaFromPassedFormula (
    ModuleInput::iterator _subformula,
    bool _ignoreOrigins = false ) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_subformula	The sub-formula to remove from the passed formula.

Parameters

_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 >.

0.15.167.4.39 excludeNotReceivedVariablesFromModel()

`void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

0.15.167.4.40 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
            const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

0.15.167.4.41 firstSubformulaToPass()

`ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.167.4.42 firstUncheckedReceivedSubformula()

`ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.167.4.43 freeSplittingVariable()

```
static void smtrat::Module::freeSplittingVariable (
            const FormulaT & _splittingVariable ) [inline], [static], [inherited]
```

0.15.167.4.44 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.167.4.45 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.167.4.46 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.167.4.47 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.167.4.48 getInfeasibleSubsets() [2/2] `std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets ( const Module & _backend ) const [protected], [inherited]`

Get the infeasible subsets the given backend provides.
Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

Parameters

_backend	The backend from which to obtain the infeasible subsets.

Returns

The infeasible subsets the given backend provides.

0.15.167.4.49 getInformationRelevantFormulas() `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.167.4.50 getModelEqualities() `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.167.4.51 getOrigins() [1/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.167.4.52 getOrigins() [2/3]

`void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.167.4.53 getOrigins() [3/3]

`const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]`

Gets the origins of the passed formula at the given position.

Parameters

_formula	The position of a formula in the passed formulas.

Returns

The origins of the passed formula at the given position.

0.15.167.4.54 getReceivedFormulaSimplified()

`std::pair< bool, FormulaT > smtrat::PModule::getReceivedFormulaSimplified ( ) [virtual], [inherited]`

Returns

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's reveived formula, if the Boolean is true.

Reimplemented from smtrat::Module.

0.15.167.4.55 hasLemmas()

`bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

0.15.167.4.56 hasValidInfeasibleSubset()

`bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]`

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.167.4.57 id()

`std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.167.4.58 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasible↩ Subsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.167.4.59 inform() `bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.
It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

Parameters

_constraint	The constraint to inform about.

Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

0.15.167.4.60 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.

0.15.167.4.61 informCore() `virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.167.4.62 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.

0.15.167.4.63 init() `void smtrat::Module::init ( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not be called twice and only before the first runBackends call.
Reimplemented in smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntBlastModule< Settings >, smtrat::FPPModule< Settings >, smtrat::CurryModule< Settings >, and smtrat::BVModule< Settings >.

0.15.167.4.64 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.167.4.65 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.

0.15.167.4.66 isPreprocessor() `bool smtrat::PModule::isPreprocessor ( ) const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.167.4.67 lemmas() `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.167.4.68 merge() `std::vector< FormulaT > smtrat::Module::merge ( const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.
({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

Parameters

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns

The vector being the two given vectors merged.

0.15.167.4.69 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.167.4.70 modelsDisjoint()

`bool smtrat::Module::modelsDisjoint ( const Model & _modelA, const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

Parameters

_modelA	The first model to check for.
_modelB	The second model to check for.

Returns

true, if there is no variable assigned by both models; false, otherwise.

0.15.167.4.71 moduleName()

`virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]`

Returns

The name of the given module type as name.

Reimplemented in smtrat::VSModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::STropModule< Settings >, smtrat::SplitSOSModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntEqModule< Settings >, smtrat::IntBlastModule< Settings >, smtrat::IncWidthModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::GBModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::CurryModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::BVModule< Settings >, and smtrat::BEModule< Settings >.

0.15.167.4.72 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

0.15.167.4.73 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.167.4.74 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.167.4.75 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.167.4.76 pPassedFormula() `const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.167.4.77 pReceivedFormula() `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.167.4.78 print() `void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.167.4.79 printAllModels() `void smtrat::Module::printAllModels ( std::ostream & _out = std::cout ) [inherited]`

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.167.4.80 printInfeasibleSubsets() `void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.167.4.81 printModel() `void smtrat::Module::printModel ( std::ostream & _out = std::cout ) const [inherited]`

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.167.4.82 printPassedFormula()

`void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.167.4.83 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.167.4.84 probablyLooping()

`bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.167.4.85 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.167.4.86 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

0.15.167.4.87 receivedVariable()

`bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

0.15.167.4.88 remove()

`void smtrat::PModule::remove ( ModuleInput::const_iterator _subformula ) [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.
However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

Parameters

_subformula	The sub formula of the received formula to remove.

Reimplemented from smtrat::Module.

0.15.167.4.89 removeCore()

```
virtual void smtrat::Module::removeCore (
    [[maybe_unused] ] 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.

0.15.167.4.90 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

0.15.167.4.91 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

0.15.167.4.92 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.167.4.93 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to this module.

0.15.167.4.94 runBackends() [1/2]

```
virtual Answer smtrat::PModule::runBackends ( ) [inline],
[virtual], [inherited]
```

Reimplemented from smtrat::Module.

0.15.167.4.95 runBackends() [2/2]

```
Answer smtrat::PModule::runBackends (
    bool _final,
    bool _full,
    carl::Variable _objective ) [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

Parameters

_final	true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT.
_full	false, if this module should avoid too expensive procedures and rather return unknown instead.
_objective	if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good.

Returns

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.167.4.96 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.

0.15.167.4.97 setThreadPriority()

```
void smtrat::Module::setThreadPriority (
    thread_priority _threadPriority ) [inline], [inherited]
```

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.167.4.98 solverState()

```
Answer smtrat::Module::solverState ( ) const [inline], [inherited]
```

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.167.4.99 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 !=.

0.15.167.4.100 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.167.4.101 updateAllModels() `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.167.4.102 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.167.4.103 updateModel() `void smtrat::PModule::updateModel ( ) const [virtual], [inherited]`

Updates the current assignment into the model.
Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.
Reimplemented from smtrat::Module.

0.15.167.4.104 usedBackends() `const std::vector<Module*>& smtrat::Module::usedBackends ( ) const [inline], [inherited]`

Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.167.5 Field Documentation

0.15.167.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.167.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.167.5.3 mBackendsFoundAnswer `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.167.5.4 mConstraintsToInform

`carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.167.5.5 mFinalCheck

`bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.167.5.6 mFirstPosInLastBranches

`std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.167.5.7 mFirstSubformulaToPass

`ModuleInput::iterator smtrat::Module::mFirstSubformulaTo↩ Pass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.167.5.8 mFirstUncheckedReceivedSubformula

`ModuleInput::const_iterator smtrat::Module↩ ::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.167.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.167.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.167.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.167.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.167.5.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std↩ ::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.167.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.167.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.167.5.16 mModelComputed `bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.167.5.17 mNumOfBranchVarsToStore `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.167.5.18 mObjectiveVariable `carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.167.5.19 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

0.15.167.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.167.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.167.5.22 mSolverState `std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.167.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

0.15.167.5.24 mUsedBackends `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.167.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.168 smtrat::subtropical::Encoding Class Reference

`#include <Subtropical.h>`

Public Member Functions

- FormulaT encode_separator (const Separator &separator, const Direction direction, const SeparatorType separator_type)

  Creates the formula for the hyperplane that linearly separates at least one variant positive frame vertex from all variant negative frame vertices.
- FormulaT encode_integer (carl::Variable var)
- Model construct_assignment (const Model &lra_model, const FormulaT &f) const

0.15.168.1 Member Function Documentation

0.15.168.1.1 construct_assignment()

```
Model smtrat::subtropical::Encoding::construct_assignment (
    const Model & lra_model,
    const FormulaT & f ) const  [inline]
```

Stores all informations retrieved from the LRA solver to construct the model

0.15.168.1.2 encode_integer()

```
FormulaT smtrat::subtropical::Encoding::encode_integer (
    carl::Variable var )  [inline]
```

0.15.168.1.3 encode_separator()

```
FormulaT smtrat::subtropical::Encoding::encode_separator (
    const Separator & separator,
    const Direction direction,
    const SeparatorType separator_type )  [inline]
```

Creates the formula for the hyperplane that linearly separates at least one variant positive frame vertex from all variant negative frame vertices.
If a weak separator is searched, the corresponding rating is included.

Parameters

separator	The separator object that stores the construction information.
negated	True, if the formula for the negated polynomial shall be constructed. False, if the formula for the original polynomial shall be constructed.

Returns

Formula that is satisfiable iff such a separating hyperplane exists.

Create the hyperplane and sign change formula
Create the rating case distinction formula
Create the strict/weak linear separating hyperplane

0.15.169 smtrat::EQGraph< VertexName > Struct Template Reference

```
#include <EQGraph.h>
```

Public Types

- using vec = boost::vecS
- using undirected = boost::undirectedS
- using vertex_property = boost::property< vertex_properties_t, VertexName >
- using Graph = boost::adjacency_list< vec, vec, undirected, vertex_property >
- using Vertex = typename boost::graph_traits< Graph >::vertex_descriptor
- using PathEdge = std::pair< VertexName, VertexName >
- using MaybePath = std::optional< std::vector< PathEdge > >

Public Member Functions

- auto properties () noexcept
- auto properties () const noexcept
- VertexName const & properties (Vertex const &v) const noexcept
- VertexName & properties (Vertex const &v) noexcept
- void add_vertex (VertexName const &name) noexcept
- void add_edge (VertexName const &a, VertexName const &b) noexcept
- void remove_edge (VertexName const &a, VertexName const &b) noexcept
- MaybePath get_path (VertexName const &a, VertexName const &b) noexcept
- MaybePath get_path (VertexName const &a, VertexName const &b, size_t bound) noexcept

Data Fields

- Graph graph
- std::map< VertexName, Vertex > vertices

0.15.169.1 Member Typedef Documentation

0.15.169.1.1 Graph `template<typename VertexName >`

```
using smtrat::EQGraph< VertexName >::Graph = boost::adjacency_list<vec, vec, undirected,
vertex_property>
```

0.15.169.1.2 MaybePath `template<typename VertexName >`

```
using smtrat::EQGraph< VertexName >::MaybePath = std::optional<std::vector<PathEdge> >
```

0.15.169.1.3 PathEdge `template<typename VertexName >`

```
using smtrat::EQGraph< VertexName >::PathEdge = std::pair<VertexName, VertexName>
```

0.15.169.1.4 undirected `template<typename VertexName >`

```
using smtrat::EQGraph< VertexName >::undirected = boost::undirectedS
```

0.15.169.1.5 vec `template<typename VertexName >`

```
using smtrat::EQGraph< VertexName >::vec = boost::vecS
```

0.15.169.1.6 Vertex `template<typename VertexName >`

```
using smtrat::EQGraph< VertexName >::Vertex = typename boost::graph_traits<Graph>::vertex_↩
descriptor
```

0.15.169.1.7 vertex_property `template<typename VertexName >`

```
using smtrat::EQGraph< VertexName >::vertex_property = boost::property<vertex_properties_t,
VertexName>
```

0.15.169.2 Member Function Documentation

0.15.169.2.1 add_edge()

```
template<typename VertexName >
void smtrat::EQGraph< VertexName >::add_edge (
            VertexName const & a,
            VertexName const & b )  [inline], [noexcept]
```

0.15.169.2.2 add_vertex()

```
template<typename VertexName >
void smtrat::EQGraph< VertexName >::add_vertex (
            VertexName const & name )  [inline], [noexcept]
```

0.15.169.2.3 get_path() [1/2]

```
template<typename VertexName >
MaybePath smtrat::EQGraph< VertexName >::get_path (
            VertexName const & a,
            VertexName const & b )  [inline], [noexcept]
```

0.15.169.2.4 get_path() [2/2]

```
template<typename VertexName >
MaybePath smtrat::EQGraph< VertexName >::get_path (
            VertexName const & a,
            VertexName const & b,
            size_t bound )  [inline], [noexcept]
```

0.15.169.2.5 properties() [1/4]

```
template<typename VertexName >
auto smtrat::EQGraph< VertexName >::properties ( ) const  [inline], [noexcept]
```

0.15.169.2.6 properties() [2/4]

```
template<typename VertexName >
auto smtrat::EQGraph< VertexName >::properties ( )  [inline], [noexcept]
```

0.15.169.2.7 properties() [3/4]

```
template<typename VertexName >
VertexName const& smtrat::EQGraph< VertexName >::properties (
            Vertex const & v ) const  [inline], [noexcept]
```

0.15.169.2.8 properties() [4/4]

```
template<typename VertexName >
VertexName& smtrat::EQGraph< VertexName >::properties (
            Vertex const & v )  [inline], [noexcept]
```

0.15.169.2.9 remove_edge()

```
template<typename VertexName >
void smtrat::EQGraph< VertexName >::remove_edge (
            VertexName const & a,
            VertexName const & b )  [inline], [noexcept]
```

0.15.169.3 Field Documentation

0.15.169.3.1 graph

```
template<typename VertexName >
Graph smtrat::EQGraph< VertexName >::graph
```

0.15.169.3.2 vertices

```
template<typename VertexName >
std::map<VertexName, Vertex> smtrat::EQGraph< VertexName >::vertices
```

0.15.170 Minisat::Equal< K > Struct Template Reference

```
#include <Map.h>
```

Public Member Functions

- bool operator() (const K &k1, const K &k2) const

0.15.170.1 Member Function Documentation

0.15.170.1.1 operator()()

```
template<class K >
bool Minisat::Equal< K >::operator() (
            const K & k1,
            const K & k2 ) const  [inline]
```

0.15.171 smtrat::parser::ErrorHandler Struct Reference

```
#include <Script.h>
```

Data Structures

- struct result

Public Member Functions

- template<typename T1 , typename T2 , typename T3 , typename T4 >
  qi::error_handler_result operator() (T1 b, T2 e, T3 where, T4 const &what) const

0.15.171.1 Member Function Documentation

0.15.171.1.1 operator()()

```
template<typename T1 , typename T2 , typename T3 , typename T4 >
qi::error_handler_result smtrat::parser::ErrorHandler::operator() (
            T1 b,
            T2 e,
            T3 where,
            T4 const & what ) const  [inline]
```

0.15.172 smtrat::ESModule< Settings > Class Template Reference

```
#include <ESModule.h>
```

Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- ESModule (const ModuleInput ∗_formula, Conditionals &_conditionals, Manager ∗_manager=NULL)
- ∼ESModule ()
- void updateModel () const

  Updates the current assignment into the model.
- Answer checkCore ()

  Checks the received formula for consistency.
- bool isPreprocessor () const
- bool appliedPreprocessing () const
- bool add (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- void remove (ModuleInput::const_iterator _subformula)

  Removes everything related to the given sub-formula of the received formula.
- Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)

  Checks the received formula for consistency.
- Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- std::pair< bool, FormulaT > getReceivedFormulaSimplified ()
- bool inform (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void deinform (const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual void init ()

  Informs all backends about the so far encountered constraints, which have not yet been communicated.
- virtual void updateAllModels ()

  Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
- virtual std::list< std::vector< carl::Variable > > getModelEqualities () const

  Partition the variables from the current model into equivalence classes according to their assigned value.
- unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const
- virtual unsigned currentlySatisfied (const FormulaT &) const
- bool receivedVariable (carl::Variable::Arg _var) const
- Answer solverState () const
- std::size_t id () const
- void setId (std::size_t _id)

  Sets this 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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.
- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)
- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

Copies the infeasible subsets of the passed formula.

- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

Stores the assignment of the current satisfiable result, if existent.

- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.172.1 Member Typedef Documentation

0.15.172.1.1 SettingsType

```
template<typename Settings >
typedef Settings smtrat::ESModule< Settings >::SettingsType
```

0.15.172.2 Member Enumeration Documentation

0.15.172.2.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

0.15.172.3 Constructor & Destructor Documentation

0.15.172.3.1 ESModule() `template<typename Settings >`

```
smtrat::ESModule< Settings >::ESModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = NULL )
```

0.15.172.3.2 ~ESModule() `template<typename Settings >`

```
smtrat::ESModule< Settings >::~ESModule ( )
```

0.15.172.4 Member Function Documentation

0.15.172.4.1 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.

0.15.172.4.2 addConstraintToInform() `void smtrat::Module::addConstraintToInform (`

```
            const FormulaT & _constraint )  [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

0.15.172.4.3 addCore() `virtual bool smtrat::Module::addCore (`

```
            [[maybe_unused] ] 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.

0.15.172.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.172.4.5 addLemma()

```
void smtrat::Module::addLemma (
    const FormulaT & _lemma,
    const LemmaType & _lt = LemmaType::NORMAL,
    const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline],
[inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.172.4.6 addOrigin()

```
void smtrat::Module::addOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.172.4.7 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.172.4.8 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula )  [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.172.4.9 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula,
            const FormulaT & _origin )  [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.172.4.10 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.

0.15.172.4.11 allModels() `const std::vector<Model>& smtrat::Module::allModels ( ) const [inline], [inherited]`

Returns

All satisfying assignments, if existent.

0.15.172.4.12 anAnswerFound() `bool smtrat::Module::anAnswerFound ( ) const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.172.4.13 answerFound() `const smtrat::Conditionals& smtrat::Module::answerFound ( ) const [inline], [inherited]`

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.172.4.14 appliedPreprocessing() `bool smtrat::PModule::appliedPreprocessing ( ) const [inline], [inherited]`

Returns

true, if the current received formula has been simplified and the result of this simplification is stored in the passed formula.

0.15.172.4.15 backendsModel() `const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.172.4.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.172.4.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.172.4.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline],
[protected], [inherited]
```

0.15.172.4.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.172.4.20 check()

```
Answer smtrat::PModule::check (
    bool _final = false,
    bool _full = true,
    carl::Variable _objective = carl::Variable::NO_VARIABLE ) [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.172.4.21 checkCore()

`template<typename Settings >`
`Answer smtrat::ESModule< Settings >::checkCore ( ) [virtual]`

Checks the received formula for consistency.

Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.172.4.22 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.

0.15.172.4.23 checkModel()

`unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]`

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.172.4.24 cleanModel()

`void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.172.4.25 clearLemmas()

`void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

0.15.172.4.26 clearModel()

`void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

0.15.172.4.27 clearModels()

`void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

0.15.172.4.28 clearPassedFormula()

`void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.172.4.29 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

0.15.172.4.30 collectOrigins() [2/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.
If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.172.4.31 collectTheoryPropagations()

`void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

0.15.172.4.32 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns

The constraints which the backends must be informed about.

0.15.172.4.33 currentlySatisfied()

```
virtual unsigned smtrat::Module::currentlySatisfied (
    const FormulaT & ) const [inline], [virtual], [inherited]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.172.4.34 currentlySatisfiedByBackend()

```
unsigned smtrat::Module::currentlySatisfiedByBackend (
    const FormulaT & _formula ) const [inherited]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.172.4.35 deinform()

```
void smtrat::Module::deinform (
    const FormulaT & _constraint ) [inherited]
```

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.172.4.36 deinformCore()

```
virtual void smtrat::Module::deinformCore (
    [[maybe_unused] ] 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.

0.15.172.4.37 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

0.15.172.4.38 eraseSubformulaFromPassedFormula()

```
ModuleInput::iterator smtrat::Module::erase↩
SubformulaFromPassedFormula (
    ModuleInput::iterator _subformula,
    bool _ignoreOrigins = false ) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_subformula	The sub-formula to remove from the passed formula.

Parameters

_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 >.

0.15.172.4.39 excludeNotReceivedVariablesFromModel()

`void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

0.15.172.4.40 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
            const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

0.15.172.4.41 firstSubformulaToPass()

`ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.172.4.42 firstUncheckedReceivedSubformula()

`ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.172.4.43 freeSplittingVariable()

```
static void smtrat::Module::freeSplittingVariable (
            const FormulaT & _splittingVariable ) [inline], [static], [inherited]
```

0.15.172.4.44 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.172.4.45 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.172.4.46 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.172.4.47 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.172.4.48 getInfeasibleSubsets() [2/2] `std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets ( const Module & _backend ) const [protected], [inherited]`

Get the infeasible subsets the given backend provides.
Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

Parameters

_backend	The backend from which to obtain the infeasible subsets.

Returns

The infeasible subsets the given backend provides.

0.15.172.4.49 getInformationRelevantFormulas() `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.172.4.50 getModelEqualities() `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.172.4.51 getOrigins() [1/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.172.4.52 getOrigins() [2/3]

`void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.172.4.53 getOrigins() [3/3]

`const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]`

Gets the origins of the passed formula at the given position.

Parameters

_formula	The position of a formula in the passed formulas.

Returns

The origins of the passed formula at the given position.

0.15.172.4.54 getReceivedFormulaSimplified()

`std::pair< bool, FormulaT > smtrat::PModule::getReceivedFormulaSimplified ( ) [virtual], [inherited]`

Returns

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's reveived formula, if the Boolean is true.

Reimplemented from smtrat::Module.

0.15.172.4.55 hasLemmas()

`bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

0.15.172.4.56 hasValidInfeasibleSubset()

`bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]`

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.172.4.57 id()

`std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.172.4.58 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasible↩ Subsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.172.4.59 inform() `bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.
It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

Parameters

_constraint	The constraint to inform about.

Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

0.15.172.4.60 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.

0.15.172.4.61 informCore() `virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.172.4.62 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.

0.15.172.4.63 init() `void smtrat::Module::init ( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not be called twice and only before the first runBackends call.
Reimplemented in smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntBlastModule< Settings >, smtrat::FPPModule< Settings >, smtrat::CurryModule< Settings >, and smtrat::BVModule< Settings >.

0.15.172.4.64 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.172.4.65 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.

0.15.172.4.66 isPreprocessor() `bool smtrat::PModule::isPreprocessor ( ) const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.172.4.67 lemmas() `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.172.4.68 merge() `std::vector< FormulaT > smtrat::Module::merge ( const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.
({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

Parameters

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns

The vector being the two given vectors merged.

0.15.172.4.69 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.172.4.70 modelsDisjoint()

`bool smtrat::Module::modelsDisjoint ( const Model & _modelA, const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

Parameters

_modelA	The first model to check for.
_modelB	The second model to check for.

Returns

true, if there is no variable assigned by both models; false, otherwise.

0.15.172.4.71 moduleName()

`virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]`

Returns

The name of the given module type as name.

Reimplemented in smtrat::VSModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::STropModule< Settings >, smtrat::SplitSOSModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntEqModule< Settings >, smtrat::IntBlastModule< Settings >, smtrat::IncWidthModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::GBModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::CurryModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::BVModule< Settings >, and smtrat::BEModule< Settings >.

0.15.172.4.72 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

0.15.172.4.73 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.172.4.74 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.172.4.75 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.172.4.76 pPassedFormula() `const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.172.4.77 pReceivedFormula() `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.172.4.78 print() `void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.172.4.79 printAllModels() `void smtrat::Module::printAllModels ( std::ostream & _out = std::cout ) [inherited]`

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.172.4.80 printInfeasibleSubsets() `void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.172.4.81 printModel() `void smtrat::Module::printModel ( std::ostream & _out = std::cout ) const [inherited]`

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.172.4.82 printPassedFormula()

`void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.172.4.83 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.172.4.84 probablyLooping()

`bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.172.4.85 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.172.4.86 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

0.15.172.4.87 receivedVariable()

`bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

0.15.172.4.88 remove()

`void smtrat::PModule::remove ( ModuleInput::const_iterator _subformula ) [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.
However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

Parameters

_subformula	The sub formula of the received formula to remove.

Reimplemented from smtrat::Module.

0.15.172.4.89 removeCore()

```
virtual void smtrat::Module::removeCore (
    [[maybe_unused] ] 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.

0.15.172.4.90 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

0.15.172.4.91 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

0.15.172.4.92 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.172.4.93 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to this module.

0.15.172.4.94 runBackends() [1/2]

```
virtual Answer smtrat::PModule::runBackends ( ) [inline],
[virtual], [inherited]
```

Reimplemented from smtrat::Module.

0.15.172.4.95 runBackends() [2/2]

```
Answer smtrat::PModule::runBackends (
    bool _final,
    bool _full,
    carl::Variable _objective ) [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

Parameters

_final	true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT.
_full	false, if this module should avoid too expensive procedures and rather return unknown instead.
_objective	if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good.

Returns

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.172.4.96 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.

0.15.172.4.97 setThreadPriority()

```
void smtrat::Module::setThreadPriority (
    thread_priority _threadPriority ) [inline], [inherited]
```

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.172.4.98 solverState()

```
Answer smtrat::Module::solverState ( ) const [inline], [inherited]
```

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.172.4.99 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 !=.

0.15.172.4.100 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.172.4.101 updateAllModels() `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.172.4.102 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.172.4.103 updateModel() `template<typename Settings > void smtrat::ESModule< Settings >::updateModel ( ) const [virtual]`

Updates the current assignment into the model.
Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.
Reimplemented from smtrat::Module.

0.15.172.4.104 usedBackends() `const std::vector<Module*>& smtrat::Module::usedBackends ( ) const [inline], [inherited]`

Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.172.5 Field Documentation

0.15.172.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.172.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.172.5.3 mBackendsFoundAnswer `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.172.5.4 mConstraintsToInform

`carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.172.5.5 mFinalCheck

`bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.172.5.6 mFirstPosInLastBranches

`std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.172.5.7 mFirstSubformulaToPass

`ModuleInput::iterator smtrat::Module::mFirstSubformulaTo↩ Pass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.172.5.8 mFirstUncheckedReceivedSubformula

`ModuleInput::const_iterator smtrat::Module↩ ::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.172.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.172.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.172.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.172.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.172.5.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std↩ ::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.172.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.172.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.172.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.172.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.172.5.18 mObjectiveVariable

`carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.172.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

0.15.172.5.20 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.172.5.21 mSmallerMusesCheckCounter

`unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.172.5.22 mSolverState

`std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.172.5.23 mTheoryPropagations

`std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

0.15.172.5.24 mUsedBackends

`std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.172.5.25 mVariableCounters

`std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.173 smtrat::ExactlyOneCommanderEncoder Class Reference

`#include <ExactlyOneCommanderEncoder.h>`

Public Member Functions

- ExactlyOneCommanderEncoder ()
- Rational encodingSize (const ConstraintT &constraint)
- bool canEncode (const ConstraintT &constraint)
- std::string name ()
- std::optional< FormulaT > encode (const ConstraintT &constraint)

  Encodes an arbitrary constraint.

Data Fields

- std::size_t problem_size

Protected Member Functions

- std::optional< FormulaT > doEncode (const ConstraintT &constraint)
- FormulaT generateVarChain (const std::set< carl::Variable > &vars, carl::FormulaType type)

0.15.173.1 Constructor & Destructor Documentation

0.15.173.1.1 ExactlyOneCommanderEncoder() `smtrat::ExactlyOneCommanderEncoder::ExactlyOne↩CommanderEncoder ( ) [inline]`

0.15.173.2 Member Function Documentation

0.15.173.2.1 canEncode() `bool smtrat::ExactlyOneCommanderEncoder::canEncode ( const ConstraintT & constraint ) [virtual]`

Implements smtrat::PseudoBoolEncoder.

0.15.173.2.2 doEncode() `std::optional< FormulaT > smtrat::ExactlyOneCommanderEncoder::do↩Encode ( const ConstraintT & constraint ) [protected], [virtual]`

Implements smtrat::PseudoBoolEncoder.

0.15.173.2.3 encode() `std::optional< FormulaT > smtrat::PseudoBoolEncoder::encode ( const ConstraintT & constraint ) [inherited]`

Encodes an arbitrary constraint.

Returns

encoded formula

0.15.173.2.4 encodingSize() `Rational smtrat::ExactlyOneCommanderEncoder::encodingSize ( const ConstraintT & constraint ) [virtual]`

Reimplemented from smtrat::PseudoBoolEncoder.

0.15.173.2.5 generateVarChain() `FormulaT smtrat::PseudoBoolEncoder::generateVarChain ( const std::set< carl::Variable > & vars, carl::FormulaType type ) [protected], [inherited]`

0.15.173.2.6 name() `std::string smtrat::ExactlyOneCommanderEncoder::name ( ) [inline], [virtual]`

Reimplemented from smtrat::PseudoBoolEncoder.

0.15.173.3 Field Documentation

0.15.173.3.1 problem_size `std::size_t smtrat::PseudoBoolEncoder::problem_size [inherited]`

0.15.174 smtrat::execution::ExecutionState Class Reference

`#include <ExecutionState.h>`

Public Member Functions

- ExecutionState ()
- void set_add_assertion_handler (std::function< void(const FormulaT &)> f)
- void set_add_soft_assertion_handler (std::function< void(const FormulaT &, Rational, const std::string &)> f)
- void set_add_objective_handler (std::function< void(const Poly &, bool)> f)
- void set_add_annotated_name_handler (std::function< void(const FormulaT &, const std::string &)> f)
- void set_remove_assertion_handler (std::function< void(const FormulaT &)> f)
- void set_remove_soft_assertion_handler (std::function< void(const FormulaT &)> f)
- void set_remove_objective_handler (std::function< void(const Poly &)> f)
- void set_remove_annotated_name_handler (std::function< void(const FormulaT &)> f)
- bool is_mode (Mode mode) const
- void set_logic (carl::Logic logic)
- auto logic () const
- void add_assertion (const FormulaT &formula)
- bool has_assertion (const FormulaT &formula) const
- const auto & assertions () const
- void add_soft_assertion (const FormulaT &formula, Rational weight, const std::string &id)
- bool has_soft_assertion (const FormulaT &formula) const
- void add_objective (const Poly &function, bool minimize)
- bool has_objective (const Poly &function)
- void push ()
- void push (size_t n)
- void pop ()
- void pop (size_t n)
- bool has_annotated_name (const std::string &n)
- bool has_annotated_name_formula (const smtrat::FormulaT &f)
- void annotate_name (const std::string &name, const smtrat::FormulaT &f)
- void enter_sat (const smtrat::Model &model, const ObjectiveValues &objectiveValues)
- void enter_unsat ()
- const smtrat::Model & model () const
- const smtrat::ObjectiveValues & objectiveValues () const
- void reset ()
- void reset_assertions ()
- void reset_answer ()

0.15.174.1 Constructor & Destructor Documentation

0.15.174.1.1 ExecutionState() `smtrat::execution::ExecutionState::ExecutionState ( ) [inline]`

0.15.174.2 Member Function Documentation

0.15.174.2.1 add_assertion()

```
void smtrat::execution::ExecutionState::add_assertion (
    const FormulaT & formula ) [inline]
```

0.15.174.2.2 add_objective()

```
void smtrat::execution::ExecutionState::add_objective (
    const Poly & function,
    bool minimize ) [inline]
```

0.15.174.2.3 add_soft_assertion()

```
void smtrat::execution::ExecutionState::add_soft_assertion (
    const FormulaT & formula,
    Rational weight,
    const std::string & id ) [inline]
```

0.15.174.2.4 annotate_name()

```
void smtrat::execution::ExecutionState::annotate_name (
    const std::string & name,
    const smtrat::FormulaT & f ) [inline]
```

0.15.174.2.5 assertions()

```
const auto& smtrat::execution::ExecutionState::assertions ( ) const
[inline]
```

0.15.174.2.6 enter_sat()

```
void smtrat::execution::ExecutionState::enter_sat (
    const smtrat::Model & model,
    const ObjectiveValues & objectiveValues ) [inline]
```

0.15.174.2.7 enter_unsat()

```
void smtrat::execution::ExecutionState::enter_unsat ( ) [inline]
```

0.15.174.2.8 has_annotated_name()

```
bool smtrat::execution::ExecutionState::has_annotated_name (
    const std::string & n ) [inline]
```

0.15.174.2.9 has_annotated_name_formula()

```
bool smtrat::execution::ExecutionState::has_annotated↩
_name_formula (
    const smtrat::FormulaT & f ) [inline]
```

0.15.174.2.10 has_assertion()

```
bool smtrat::execution::ExecutionState::has_assertion (
    const FormulaT & formula ) const [inline]
```

0.15.174.2.11 has_objective()

```
bool smtrat::execution::ExecutionState::has_objective (
    const Poly & function ) [inline]
```

0.15.174.2.12 has_soft_assertion()

```
bool smtrat::execution::ExecutionState::has_soft_assertion (
    const FormulaT & formula ) const [inline]
```

0.15.174.2.13 is_mode()

`bool smtrat::execution::ExecutionState::is_mode ( Mode mode ) const [inline]`

0.15.174.2.14 logic()

`auto smtrat::execution::ExecutionState::logic ( ) const [inline]`

0.15.174.2.15 model()

`const smtrat::Model& smtrat::execution::ExecutionState::model ( ) const [inline]`

0.15.174.2.16 objectiveValues()

`const smtrat::ObjectiveValues& smtrat::execution::ExecutionState::objectiveValues ( ) const [inline]`

0.15.174.2.17 pop() [1/2]

`void smtrat::execution::ExecutionState::pop ( ) [inline]`

0.15.174.2.18 pop() [2/2]

`void smtrat::execution::ExecutionState::pop ( size_t n ) [inline]`

0.15.174.2.19 push() [1/2]

`void smtrat::execution::ExecutionState::push ( ) [inline]`

0.15.174.2.20 push() [2/2]

`void smtrat::execution::ExecutionState::push ( size_t n ) [inline]`

0.15.174.2.21 reset()

`void smtrat::execution::ExecutionState::reset ( ) [inline]`

0.15.174.2.22 reset_answer()

`void smtrat::execution::ExecutionState::reset_answer ( ) [inline]`

0.15.174.2.23 reset_assertions()

`void smtrat::execution::ExecutionState::reset_assertions ( ) [inline]`

0.15.174.2.24 set_add_annotated_name_handler()

`void smtrat::execution::ExecutionState::set_add_annotated_name_handler ( std::function< void(const FormulaT &, const std::string &)> f ) [inline]`

0.15.174.2.25 set_add_assertion_handler()

`void smtrat::execution::ExecutionState::set_add_assertion_handler ( std::function< void(const FormulaT &)> f ) [inline]`

0.15.174.2.26 set_add_objective_handler()

`void smtrat::execution::ExecutionState::set_add_objective_handler ( std::function< void(const Poly &, bool)> f ) [inline]`

0.15.174.2.27 set_add_soft_assertion_handler()

```
void smtrat::execution::ExecutionState::set_add↩
_soft_assertion_handler (
            std::function< void(const FormulaT &, Rational, const std::string &)> f )  [inline]
```

0.15.174.2.28 set_logic()

```
void smtrat::execution::ExecutionState::set_logic (
            carl::Logic logic )  [inline]
```

0.15.174.2.29 set_remove_annotated_name_handler()

```
void smtrat::execution::ExecutionState↩
::set_remove_annotated_name_handler (
            std::function< void(const FormulaT &)> f )  [inline]
```

0.15.174.2.30 set_remove_assertion_handler()

```
void smtrat::execution::ExecutionState::set_↩
remove_assertion_handler (
            std::function< void(const FormulaT &)> f )  [inline]
```

0.15.174.2.31 set_remove_objective_handler()

```
void smtrat::execution::ExecutionState::set_↩
remove_objective_handler (
            std::function< void(const Poly &)> f )  [inline]
```

0.15.174.2.32 set_remove_soft_assertion_handler()

```
void smtrat::execution::ExecutionState::set_↩
remove_soft_assertion_handler (
            std::function< void(const FormulaT &)> f )  [inline]
```

0.15.175 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

0.15.175.1 Member Typedef Documentation

0.15.175.1.1 Value `typedef types::AttributeValue smtrat::parser::InstructionHandler::Value [inherited]`

0.15.175.2 Constructor & Destructor Documentation

0.15.175.2.1 Executor() template<typename Strategy >

```
smtrat::Executor< Strategy >::Executor (
            Strategy & solver )  [inline]
```

0.15.175.2.2 ~Executor() template<typename Strategy >

```
smtrat::Executor< Strategy >::~Executor ( )  [inline]
```

0.15.175.3 Member Function Documentation

0.15.175.3.1 add() template<typename Strategy >

```
void smtrat::Executor< Strategy >::add (
            const smtrat::FormulaT & f )  [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

0.15.175.3.2 addInstruction() void smtrat::parser::InstructionHandler::addInstruction (

```
            std::function< void()> bind )  [inline], [inherited]
```

0.15.175.3.3 addObjective() template<typename Strategy >

```
void smtrat::Executor< Strategy >::addObjective (
            const smtrat::Poly & p,
            smtrat::parser::OptimizationType ot )  [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

0.15.175.3.4 addSoft() template<typename Strategy >

```
void smtrat::Executor< Strategy >::addSoft (
            const smtrat::FormulaT & f,
            smtrat::Rational weight,
            const std::string & id )  [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

0.15.175.3.5 annotateName() template<typename Strategy >

```
void smtrat::Executor< Strategy >::annotateName (
            const smtrat::FormulaT & f,
            const std::string & name )  [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

0.15.175.3.6 check() template<typename Strategy >

```
void smtrat::Executor< Strategy >::check ( )  [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

0.15.175.3.7 cleanModel() void smtrat::parser::InstructionHandler::cleanModel (

```
            smtrat::Model & model ) const  [inline], [inherited]
```

0.15.175.3.8 declareFun()

```
template<typename Strategy >
void smtrat::Executor< Strategy >::declareFun (
            const carl::Variable & ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

0.15.175.3.9 declareSort()

```
template<typename Strategy >
void smtrat::Executor< Strategy >::declareSort (
            const std::string & ,
            const unsigned & ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

0.15.175.3.10 defineSort()

```
template<typename Strategy >
void smtrat::Executor< Strategy >::defineSort (
            const std::string & ,
            const std::vector< std::string > & ,
            const carl::Sort & ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

0.15.175.3.11 diagnostic() [1/2]

```
std::ostream& smtrat::parser::InstructionHandler::diagnostic (
) [inline], [inherited]
```

0.15.175.3.12 diagnostic() [2/2]

```
void smtrat::parser::InstructionHandler::diagnostic (
            const std::string & s ) [inline], [inherited]
```

0.15.175.3.13 echo()

```
virtual void smtrat::parser::InstructionHandler::echo (
            const std::string & s ) [inline], [virtual], [inherited]
```

0.15.175.3.14 eliminateQuantifiers()

```
template<typename Strategy >
void smtrat::Executor< Strategy >::eliminateQuantifiers (
            const smtrat::qe::QEQuery & ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

0.15.175.3.15 error()

```
OutputWrapper smtrat::parser::InstructionHandler::error ( ) [inline],
[inherited]
```

0.15.175.3.16 exit()

```
template<typename Strategy >
void smtrat::Executor< Strategy >::exit ( ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

0.15.175.3.17 get_info()

```
const auto& smtrat::parser::InstructionHandler::get_info (
            const std::string & key ) const [inline], [inherited]
```

0.15.175.3.18 getAllModels()

```
template<typename Strategy >
void smtrat::Executor< Strategy >::getAllModels ( ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

0.15.175.3.19 getAssertions()

`template<typename Strategy >`
`void smtrat::Executor< Strategy >::getAssertions ( ) [inline], [virtual]`
Implements smtrat::parser::InstructionHandler.

0.15.175.3.20 getAssignment()

`template<typename Strategy >`
`void smtrat::Executor< Strategy >::getAssignment ( ) [inline], [virtual]`
Implements smtrat::parser::InstructionHandler.

0.15.175.3.21 getExitCode()

`template<typename Strategy >`
`int smtrat::Executor< Strategy >::getExitCode ( ) const [inline]`

0.15.175.3.22 getInfo()

`void smtrat::parser::InstructionHandler::getInfo ( const std::string & key ) [inline], [inherited]`

0.15.175.3.23 getModel()

`template<typename Strategy >`
`void smtrat::Executor< Strategy >::getModel ( ) [inline], [virtual]`
Implements smtrat::parser::InstructionHandler.

0.15.175.3.24 getObjectives()

`template<typename Strategy >`
`void smtrat::Executor< Strategy >::getObjectives ( ) [inline], [virtual]`
Implements smtrat::parser::InstructionHandler.

0.15.175.3.25 getOption()

`void smtrat::parser::InstructionHandler::getOption ( const std::string & key ) [inline], [inherited]`

0.15.175.3.26 getProof()

`template<typename Strategy >`
`void smtrat::Executor< Strategy >::getProof ( ) [inline], [virtual]`
Implements smtrat::parser::InstructionHandler.

0.15.175.3.27 getUnsatCore()

`template<typename Strategy >`
`void smtrat::Executor< Strategy >::getUnsatCore ( ) [inline], [virtual]`
Implements smtrat::parser::InstructionHandler.

0.15.175.3.28 getValue()

`template<typename Strategy >`
`void smtrat::Executor< Strategy >::getValue ( const std::vector< carl::Variable > & ) [inline], [virtual]`
Implements smtrat::parser::InstructionHandler.

0.15.175.3.29 has_info()

`bool smtrat::parser::InstructionHandler::has_info ( const std::string & key ) const [inline], [inherited]`

0.15.175.3.30 hasInstructions()

`bool smtrat::parser::InstructionHandler::hasInstructions ( ) const [inline], [inherited]`

0.15.175.3.31 info()

```
OutputWrapper smtrat::parser::InstructionHandler::info ( ) [inline], [inherited]
```

0.15.175.3.32 option()

```
template<typename T >
T smtrat::parser::InstructionHandler::option (
            const std::string & key ) const [inline], [inherited]
```

0.15.175.3.33 pop()

```
template<typename Strategy >
void smtrat::Executor< Strategy >::pop (
            std::size_t n ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

0.15.175.3.34 printInstruction()

```
bool smtrat::parser::InstructionHandler::printInstruction ( )
const [inline], [inherited]
```

0.15.175.3.35 push()

```
template<typename Strategy >
void smtrat::Executor< Strategy >::push (
            std::size_t n ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

0.15.175.3.36 regular() [1/2]

```
std::ostream& smtrat::parser::InstructionHandler::regular ( )
[inline], [inherited]
```

0.15.175.3.37 regular() [2/2]

```
void smtrat::parser::InstructionHandler::regular (
            const std::string & s ) [inline], [inherited]
```

0.15.175.3.38 reset()

```
template<typename Strategy >
void smtrat::Executor< Strategy >::reset ( ) [inline], [virtual]
```

Reimplemented from smtrat::parser::InstructionHandler.

0.15.175.3.39 resetAssertions()

```
template<typename Strategy >
void smtrat::Executor< Strategy >::resetAssertions ( ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

0.15.175.3.40 runInstructions()

```
void smtrat::parser::InstructionHandler::runInstructions ( )
[inline], [inherited]
```

0.15.175.3.41 setArtificialVariables()

```
void smtrat::parser::InstructionHandler::setArtificial↩
Variables (
            std::vector< smtrat::ModelVariable > && vars ) [inline], [inherited]
```

0.15.175.3.42 setInfo()

```
void smtrat::parser::InstructionHandler::setInfo (
            const Attribute & attr ) [inline], [inherited]
```

0.15.175.3.43 setLogic()

```
template<typename Strategy >
void smtrat::Executor< Strategy >::setLogic (
            const carl::Logic & logic ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

0.15.175.3.44 setOption()

```
void smtrat::parser::InstructionHandler::setOption (
            const Attribute & option ) [inline], [inherited]
```

0.15.175.3.45 setStream()

```
void smtrat::parser::InstructionHandler::setStream (
            const std::string & s,
            std::ostream & os ) [inline], [protected], [inherited]
```

0.15.175.3.46 warn()

```
OutputWrapper smtrat::parser::InstructionHandler::warn ( ) [inline],
[inherited]
```

0.15.175.4 Field Documentation

0.15.175.4.1 infos

```
VariantMap<std::string, Value> smtrat::parser::InstructionHandler::infos
[protected], [inherited]
```

0.15.175.4.2 lastAnswer

```
template<typename Strategy >
smtrat::Answer smtrat::Executor< Strategy >::lastAnswer
```

0.15.175.4.3 mDiagnostic

```
std::ostream smtrat::parser::InstructionHandler::mDiagnostic [protected],
[inherited]
```

0.15.175.4.4 mRegular

```
std::ostream smtrat::parser::InstructionHandler::mRegular [protected],
[inherited]
```

0.15.175.4.5 options

```
VariantMap<std::string, Value> smtrat::parser::InstructionHandler::options
[protected], [inherited]
```

0.15.175.4.6 streams

```
std::map<std::string, std::ofstream> smtrat::parser::InstructionHandler↩
::streams [protected], [inherited]
```

0.15.176 smtrat::covering_ng::formula::ExhaustiveImplicants Class Reference

```
#include <FormulaEvaluation.h>
```

Public Types

- using ImplicantComplexityOrdering = std::function< bool(const boost::container::flat_set< cadcells::Constraint > &, const boost::container::flat_set< cadcells::Constraint > &)>

Public Member Functions

- ExhaustiveImplicants (ImplicantComplexityOrdering implicant_complexity_ordering)
- ExhaustiveImplicants (ImplicantComplexityOrdering implicant_complexity_ordering, std::size_t results)
- ExhaustiveImplicants (ImplicantComplexityOrdering implicant_complexity_ordering, std::size_t results, std↩ ::size_t pruning)
- void set_formula (typename cadcells::Polynomial::ContextType c, const FormulaT &f)

  ExhaustiveImplicants.
- void extend_valuation (const cadcells::Assignment &ass)
- void revert_valuation (const cadcells::Assignment &ass)
- std::vector< boost::container::flat_set< cadcells::Constraint > > compute_implicants () const
- Valuation root_valuation () const

0.15.176.1 Member Typedef Documentation

0.15.176.1.1 ImplicantComplexityOrdering

```
using smtrat::covering_ng::formula::ExhaustiveImplicants::ImplicantCom
= std::function<bool(const boost::container::flat_set<cadcells::Constraint>&, const boost↩
::container::flat_set<cadcells::Constraint>&)>
```

0.15.176.2 Constructor & Destructor Documentation

0.15.176.2.1 ExhaustiveImplicants() [1/3]

```
smtrat::covering_ng::formula::ExhaustiveImplicants::↩
ExhaustiveImplicants (
            ImplicantComplexityOrdering implicant_complexity_ordering )  [inline]
```

0.15.176.2.2 ExhaustiveImplicants() [2/3]

```
smtrat::covering_ng::formula::ExhaustiveImplicants::↩
ExhaustiveImplicants (
            ImplicantComplexityOrdering implicant_complexity_ordering,
            std::size_t results )  [inline]
```

0.15.176.2.3 ExhaustiveImplicants() [3/3]

```
smtrat::covering_ng::formula::ExhaustiveImplicants::↩
ExhaustiveImplicants (
            ImplicantComplexityOrdering implicant_complexity_ordering,
            std::size_t results,
            std::size_t pruning )  [inline]
```

0.15.176.3 Member Function Documentation

0.15.176.3.1 compute_implicants()

```
std::vector< boost::container::flat_set< cadcells::Constraint
> > smtrat::covering_ng::formula::ExhaustiveImplicants::compute_implicants ( ) const
```

0.15.176.3.2 extend_valuation()

```
void smtrat::covering_ng::formula::ExhaustiveImplicants::extend↩
_valuation (
            const cadcells::Assignment & ass )
```

0.15.176.3.3 revert_valuation()

```
void smtrat::covering_ng::formula::ExhaustiveImplicants::revert_valuation (
            const cadcells::Assignment & ass )
```

0.15.176.3.4 root_valuation()

```
Valuation smtrat::covering_ng::formula::ExhaustiveImplicants::root_valuation ( ) const
```

0.15.176.3.5 set_formula()

```
void smtrat::covering_ng::formula::ExhaustiveImplicants::set_formula (
            typename cadcells::Polynomial::ContextType c,
            const FormulaT & f )
```

ExhaustiveImplicants.

0.15.177 smtrat::mcsat::fm::Explanation< Settings > Struct Template Reference

```
#include <Explanation.h>
```

Public Member Functions

- std::optional< mcsat::Explanation > operator() (const mcsat::Bookkeeping &data, carl::Variable var, const FormulasT &reason, bool force_use_core) const

0.15.177.1 Member Function Documentation

0.15.177.1.1 operator()()

```
template<class Settings >
std::optional< mcsat::Explanation > smtrat::mcsat::fm::Explanation< Settings >::operator() (
            const mcsat::Bookkeeping & data,
            carl::Variable var,
            const FormulasT & reason,
            bool force_use_core ) const
```

0.15.178 smtrat::mcsat::icp::Explanation Struct Reference

```
#include <Explanation.h>
```

Public Member Functions

- std::optional< mcsat::Explanation > operator() (const mcsat::Bookkeeping &data, carl::Variable var, const FormulasT &reason, bool force_use_core) const

0.15.178.1 Member Function Documentation

0.15.178.1.1 operator()()

```
std::optional< mcsat::Explanation > smtrat::mcsat::icp::Explanation::operator() (
            const mcsat::Bookkeeping & data,
            carl::Variable var,
            const FormulasT & reason,
            bool force_use_core ) const
```

0.15.179 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.

0.15.179.1 Member Function Documentation

0.15.179.1.1 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)

0.15.180 smtrat::mcsat::onecell::Explanation Struct Reference

```
#include <Explanation.h>
```

Public Member Functions

- std::optional< mcsat::Explanation > operator() (const mcsat::Bookkeeping &trail, carl::Variable var, const FormulasT &reason, bool) const

0.15.180.1 Member Function Documentation

0.15.180.1.1 operator()()

```
std::optional< mcsat::Explanation > smtrat::mcsat::onecell::Explanation↩
::operator() (
            const mcsat::Bookkeeping & trail,
            carl::Variable var,
            const FormulasT & reason,
            bool ) const
```

0.15.181 smtrat::mcsat::onecellcad::levelwise::Explanation< Setting1, Setting2 > Struct Template Reference

`#include <Explanation.h>`

Public Member Functions

- std::optional< mcsat::Explanation > operator() (const mcsat::Bookkeeping &trail, carl::Variable var, const FormulasT &trailLiterals, bool) const

0.15.181.1 Member Function Documentation

0.15.181.1.1 operator()()

```
template<class Setting1 , class Setting2 >
std::optional< mcsat::Explanation > smtrat::mcsat::onecellcad::levelwise::Explanation< Setting1,
Setting2 >::operator() (
            const mcsat::Bookkeeping & trail,
            carl::Variable var,
            const FormulasT & trailLiterals,
            bool  ) const
```

0.15.182 smtrat::mcsat::onecellcad::recursive::Explanation< Setting1, Setting2 > Struct Template Reference

`#include <Explanation.h>`

Public Member Functions

- std::optional< mcsat::Explanation > operator() (const mcsat::Bookkeeping &trail, carl::Variable var, const FormulasT &trailLiterals, bool) const

0.15.182.1 Member Function Documentation

0.15.182.1.1 operator()()

```
template<class Setting1 , class Setting2 >
std::optional< mcsat::Explanation > smtrat::mcsat::onecellcad::recursive::Explanation< Setting1,
Setting2 >::operator() (
            const mcsat::Bookkeeping & trail,
            carl::Variable var,
            const FormulasT & trailLiterals,
            bool  ) const
```

0.15.183 smtrat::mcsat::vs::Explanation Struct Reference

`#include <Explanation.h>`

Public Member Functions

- std::optional< mcsat::Explanation > operator() (const mcsat::Bookkeeping &data, carl::Variable var, const FormulasT &reason, bool) const

0.15.183.1 Member Function Documentation

0.15.183.1.1 operator()()

```
std::optional< mcsat::Explanation > smtrat::mcsat::vs::Explanation↩
::operator() (
            const mcsat::Bookkeeping & data,
            carl::Variable var,
            const FormulasT & reason,
            bool  ) const
```

0.15.184 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.

0.15.184.1 Constructor & Destructor Documentation

0.15.184.1.1 ExplanationGenerator()

```
smtrat::mcsat::nlsat::ExplanationGenerator::Explanation↩
Generator (
            const std::vector< FormulaT > & constraints,
            const std::vector< carl::Variable > & vars,
            carl::Variable ,
            const Model & model )  [inline]
```

0.15.184.2 Member Function Documentation

0.15.184.2.1 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.

0.15.184.2.2 getExplanation()

```
mcsat::Explanation smtrat::mcsat::nlsat::ExplanationGenerator↩
::getExplanation (
            const FormulaT & impliedAtom ) const  [inline]
```

Compute explanation in clausal form.

0.15.185 smtrat::mcsat::vs::ExplanationGenerator< Settings > Class Template Reference

```
#include <ExplanationGenerator.h>
```

Public Member Functions

- ExplanationGenerator (const std::vector< FormulaT > &constraints, const std::vector< carl::Variable > &variableOrdering, const carl::Variable &targetVar, const Model &model)
- std::optional< mcsat::Explanation > getExplanation () const

0.15.185.1 Constructor & Destructor Documentation

0.15.185.1.1 ExplanationGenerator()

```
template<class Settings >
smtrat::mcsat::vs::ExplanationGenerator< Settings >::ExplanationGenerator (
            const std::vector< FormulaT > & constraints,
            const std::vector< carl::Variable > & variableOrdering,
            const carl::Variable & targetVar,
            const Model & model )  [inline]
```

0.15.185.2 Member Function Documentation

0.15.185.2.1 getExplanation()

```
template<class Settings >
std::optional<mcsat::Explanation> smtrat::mcsat::vs::ExplanationGenerator< Settings >::get↩
Explanation ( ) const  [inline]
```

0.15.186 smtrat::expression::Expression Class Reference

```
#include <Expression.h>
```

Public Member Functions

- Expression (carl::Variable::Arg var)
- Expression (ITEType _type, Expression &&_if, Expression &&_then, Expression &&_else)
- Expression (QuantifierType _type, std::vector< carl::Variable > &&_variables, Expression &&_expression)
- Expression (UnaryType _type, Expression &&_expression)
- Expression (UnaryType _type, const Expression &_expression)
- Expression (BinaryType _type, Expression &&_lhs, Expression &&_rhs)
- Expression (NaryType _type, Expressions &&_expressions)
- Expression (NaryType _type, const std::initializer_list< Expression > &_expressions)
- const ExpressionContent & getContent () const
- const ExpressionContent * getContentPtr () const
- const ExpressionContent * getNegationPtr () const
- bool isITE () const
- const ITEExpression & getITE () const
- bool isQuantifier () const
- const QuantifierExpression & getQuantifier () const
- bool isUnary () const
- const UnaryExpression & getUnary () const
- bool isBinary () const
- const BinaryExpression & getBinary () const
- bool is_nary () const
- const NaryExpression & getNary () const
- bool operator== (const Expression &expr) const
- bool operator< (const Expression &expr) const
- bool operator!= (const Expression &expr) const

Friends

- struct std::hash< smtrat::expression::Expression >
- class ExpressionPool
- class ExpressionModifier

0.15.186.1 Constructor & Destructor Documentation

0.15.186.1.1 Expression() [1/8]

```
smtrat::expression::Expression::Expression (
carl::Variable::Arg var ) [explicit]
```

0.15.186.1.2 Expression() [2/8]

```
smtrat::expression::Expression::Expression (
ITEType _type,
Expression && _if,
Expression && _then,
Expression && _else ) [explicit]
```

0.15.186.1.3 Expression() [3/8]

```
smtrat::expression::Expression::Expression (
QuantifierType _type,
std::vector< carl::Variable > && _variables,
Expression && _expression ) [explicit]
```

0.15.186.1.4 Expression() [4/8]

```
smtrat::expression::Expression::Expression (
UnaryType _type,
Expression && _expression ) [explicit]
```

0.15.186.1.5 Expression() [5/8]

```
smtrat::expression::Expression::Expression (
UnaryType _type,
const Expression & _expression ) [inline], [explicit]
```

0.15.186.1.6 Expression() [6/8]

```
smtrat::expression::Expression::Expression (
BinaryType _type,
Expression && _lhs,
Expression && _rhs ) [explicit]
```

0.15.186.1.7 Expression() [7/8]

```
smtrat::expression::Expression::Expression (
NaryType _type,
Expressions && _expressions ) [explicit]
```

0.15.186.1.8 Expression() [8/8]

```
smtrat::expression::Expression::Expression (
NaryType _type,
const std::initializer_list< Expression > & _expressions ) [explicit]
```

0.15.186.2 Member Function Documentation

0.15.186.2.1 getBinary() `const BinaryExpression& smtrat::expression::Expression::getBinary ( ) const`

0.15.186.2.2 getContent() `const ExpressionContent& smtrat::expression::Expression::getContent ( ) const [inline]`

0.15.186.2.3 getContentPtr() `const ExpressionContent* smtrat::expression::Expression::getContentPtr ( ) const [inline]`

0.15.186.2.4 getITE() `const ITEExpression& smtrat::expression::Expression::getITE ( ) const`

0.15.186.2.5 getNary() `const NaryExpression& smtrat::expression::Expression::getNary ( ) const`

0.15.186.2.6 getNegationPtr() `const ExpressionContent* smtrat::expression::Expression::getNegationPtr ( ) const`

0.15.186.2.7 getQuantifier() `const QuantifierExpression& smtrat::expression::Expression::getQuantifier ( ) const`

0.15.186.2.8 getUnary() `const UnaryExpression& smtrat::expression::Expression::getUnary ( ) const`

0.15.186.2.9 is_nary() `bool smtrat::expression::Expression::is_nary ( ) const`

0.15.186.2.10 isBinary() `bool smtrat::expression::Expression::isBinary ( ) const`

0.15.186.2.11 isITE() `bool smtrat::expression::Expression::isITE ( ) const`

0.15.186.2.12 isQuantifier() `bool smtrat::expression::Expression::isQuantifier ( ) const`

0.15.186.2.13 isUnary() `bool smtrat::expression::Expression::isUnary ( ) const`

0.15.186.2.14 operator"!=() `bool smtrat::expression::Expression::operator!= ( const Expression & expr ) const`

0.15.186.2.15 operator<() `bool smtrat::expression::Expression::operator< ( const Expression & expr ) const`

0.15.186.2.16 operator==() `bool smtrat::expression::Expression::operator== ( const Expression & expr ) const`

0.15.186.3 Friends And Related Function Documentation

0.15.186.3.1 ExpressionModifier `friend class ExpressionModifier [friend]`

0.15.186.3.2 ExpressionPool `friend class ExpressionPool [friend]`

0.15.186.3.3 std::hash< smtrat::expression::Expression > `friend struct std::hash< smtrat::expression::Expression > [friend]`

0.15.187 smtrat::expression::ExpressionContent Struct Reference

`#include <ExpressionContent.h>`

Public Types

- typedef boost::variant< carl::Variable, ITEExpression, QuantifierExpression, UnaryExpression, BinaryExpression, NaryExpression > Content

Public Member Functions

- template<typename... T>
  ExpressionContent (T &&... t)

Data Fields

- Content content
- std::size_t id
- std::size_t hash
- const ExpressionContent * negation

Friends

- struct std::hash< Content >

0.15.187.1 Member Typedef Documentation

0.15.187.1.1 Content `typedef boost::variant< carl::Variable, ITEExpression, QuantifierExpression, UnaryExpression, BinaryExpression, NaryExpression > smtrat::expression::ExpressionContent::Content`

0.15.187.2 Constructor & Destructor Documentation

0.15.187.2.1 ExpressionContent() `template<typename... T> smtrat::expression::ExpressionContent::ExpressionContent ( T &&... t ) [inline]`

0.15.187.3 Friends And Related Function Documentation

0.15.187.3.1 std::hash< Content > `friend struct std::hash< Content > [friend]`

0.15.187.4 Field Documentation

0.15.187.4.1 content `Content smtrat::expression::ExpressionContent::content`

0.15.187.4.2 hash `std::size_t smtrat::expression::ExpressionContent::hash`

0.15.187.4.3 id `std::size_t smtrat::expression::ExpressionContent::id`

0.15.187.4.4 negation `const ExpressionContent* smtrat::expression::ExpressionContent::negation`

0.15.188 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

0.15.188.1 Member Function Documentation

0.15.188.1.1 convert() `FormulaT smtrat::expression::ExpressionConverter::convert ( const Expression & expr ) [inline], [protected]`

0.15.188.1.2 operator()() [1/7] `FormulaT smtrat::expression::ExpressionConverter::operator() ( carl::Variable::Arg var ) [inline]`

0.15.188.1.3 operator()() [2/7] `FormulaT smtrat::expression::ExpressionConverter::operator() ( const BinaryExpression & expr ) [inline]`

0.15.188.1.4 operator()() [3/7] `FormulaT smtrat::expression::ExpressionConverter::operator() ( const Expression & expr ) [inline]`

0.15.188.1.5 operator()() [4/7]

`FormulaT smtrat::expression::ExpressionConverter::operator() ( const ITEExpression & expr ) [inline]`

0.15.188.1.6 operator()() [5/7]

`FormulaT smtrat::expression::ExpressionConverter::operator() ( const NaryExpression & expr ) [inline]`

0.15.188.1.7 operator()() [6/7]

`FormulaT smtrat::expression::ExpressionConverter::operator() ( const QuantifierExpression & expr ) [inline]`

0.15.188.1.8 operator()() [7/7]

`FormulaT smtrat::expression::ExpressionConverter::operator() ( const UnaryExpression & expr ) [inline]`

0.15.188.2 Field Documentation

0.15.188.2.1 mGlobalFormulas

`FormulasT smtrat::expression::ExpressionConverter::mGlobalFormulas [protected]`

0.15.189 smtrat::expression::ExpressionModifier Class Reference

`#include <ExpressionVisitor.h>`

Public Types

- typedef std::function< const ExpressionContent *(const Expression &)> VisitorFunction

Public Member Functions

- void setPre (const VisitorFunction &f)
- void setPost (const VisitorFunction &f)
- Expression visit (const Expression &expression)
- const ExpressionContent * operator() (carl::Variable::Arg)
- const ExpressionContent * operator() (const ITEExpression &expr)
- const ExpressionContent * operator() (const QuantifierExpression &expr)
- const ExpressionContent * operator() (const UnaryExpression &expr)
- const ExpressionContent * operator() (const BinaryExpression &expr)
- const ExpressionContent * operator() (const NaryExpression &expr)

0.15.189.1 Member Typedef Documentation

0.15.189.1.1 VisitorFunction

`typedef std::function<const ExpressionContent*(const Expression&)> smtrat::expression::ExpressionModifier::VisitorFunction`

0.15.189.2 Member Function Documentation

0.15.189.2.1 operator()() [1/6]

`const ExpressionContent* smtrat::expression::ExpressionModifier::operator() ( carl::Variable::Arg ) [inline]`

0.15.189.2.2 operator()() [2/6]

```
const ExpressionContent* smtrat::expression::ExpressionModifier↩
::operator() (
    const BinaryExpression & expr ) [inline]
```

0.15.189.2.3 operator()() [3/6]

```
const ExpressionContent* smtrat::expression::ExpressionModifier↩
::operator() (
    const ITEExpression & expr ) [inline]
```

0.15.189.2.4 operator()() [4/6]

```
const ExpressionContent* smtrat::expression::ExpressionModifier↩
::operator() (
    const NaryExpression & expr ) [inline]
```

0.15.189.2.5 operator()() [5/6]

```
const ExpressionContent* smtrat::expression::ExpressionModifier↩
::operator() (
    const QuantifierExpression & expr ) [inline]
```

0.15.189.2.6 operator()() [6/6]

```
const ExpressionContent* smtrat::expression::ExpressionModifier↩
::operator() (
    const UnaryExpression & expr ) [inline]
```

0.15.189.2.7 setPost()

```
void smtrat::expression::ExpressionModifier::setPost (
    const VisitorFunction & f ) [inline]
```

0.15.189.2.8 setPre()

```
void smtrat::expression::ExpressionModifier::setPre (
    const VisitorFunction & f ) [inline]
```

0.15.189.2.9 visit()

```
Expression smtrat::expression::ExpressionModifier::visit (
    const Expression & expression ) [inline]
```

0.15.190 smtrat::expression::ExpressionPool Class Reference

`#include <ExpressionPool.h>`

Public Member Functions

- const ExpressionContent * create (carl::Variable::Arg var)
- const ExpressionContent * create (ITEType _type, Expression &&_if, Expression &&_then, Expression &&↩ _else)
- const ExpressionContent * create (QuantifierType _type, std::vector< carl::Variable > &&_variables, Expression &&_expression)
- const ExpressionContent * create (UnaryType _type, Expression &&_expression)
- const ExpressionContent * create (BinaryType _type, Expression &&_lhs, Expression &&_rhs)
- const ExpressionContent * create (NaryType _type, Expressions &&_expressions)
- const ExpressionContent * create (NaryType _type, const std::initializer_list< Expression > &_expressions)

Static Public Member Functions

- template<typename... Args>
  static const ExpressionContent * create (Args &&... args)

Protected Member Functions

- ExpressionPool ()
- const ExpressionContent * add (ExpressionContent *_ec)

Friends

- class ExpressionModifier

0.15.190.1 Constructor & Destructor Documentation

0.15.190.1.1 ExpressionPool() `smtrat::expression::ExpressionPool::ExpressionPool ( ) [inline], [protected]`

0.15.190.2 Member Function Documentation

0.15.190.2.1 add() `const ExpressionContent* smtrat::expression::ExpressionPool::add ( ExpressionContent * _ec ) [protected]`

0.15.190.2.2 create() [1/8] `template<typename... Args> static const ExpressionContent* smtrat::expression::ExpressionPool::create ( Args &&... args ) [inline], [static]`

0.15.190.2.3 create() [2/8] `const ExpressionContent* smtrat::expression::ExpressionPool::create ( BinaryType _type, Expression && _lhs, Expression && _rhs )`

0.15.190.2.4 create() [3/8] `const ExpressionContent* smtrat::expression::ExpressionPool::create ( carl::Variable::Arg var )`

0.15.190.2.5 create() [4/8] `const ExpressionContent* smtrat::expression::ExpressionPool::create ( ITEType _type, Expression && _if, Expression && _then, Expression && _else )`

0.15.190.2.6 create() [5/8] `const ExpressionContent* smtrat::expression::ExpressionPool::create ( NaryType _type, const std::initializer_list< Expression > & _expressions )`

0.15.190.2.7 create() [6/8]

```
const ExpressionContent* smtrat::expression::ExpressionPool::create
(
            NaryType _type,
            Expressions && _expressions )
```

0.15.190.2.8 create() [7/8]

```
const ExpressionContent* smtrat::expression::ExpressionPool::create
(
            QuantifierType _type,
            std::vector< carl::Variable > && _variables,
            Expression && _expression )
```

0.15.190.2.9 create() [8/8]

```
const ExpressionContent* smtrat::expression::ExpressionPool::create
(
            UnaryType _type,
            Expression && _expression )
```

0.15.190.3 Friends And Related Function Documentation

0.15.190.3.1 ExpressionModifier

```
friend class ExpressionModifier [friend]
```

0.15.191 smtrat::parser::ParserState::ExpressionScope Struct Reference

```
#include <ParserState.h>
```

Public Member Functions

- ExpressionScope (const ParserState &state)
- void discharge (ParserState &state)

0.15.191.1 Constructor & Destructor Documentation

0.15.191.1.1 ExpressionScope()

```
smtrat::parser::ParserState::ExpressionScope::ExpressionScope (
            const ParserState & state ) [inline]
```

0.15.191.2 Member Function Documentation

0.15.191.2.1 discharge()

```
void smtrat::parser::ParserState::ExpressionScope::discharge (
            ParserState & state ) [inline]
```

0.15.192 smtrat::expression::ExpressionTypeChecker< T > Struct Template Reference

```
#include <ExpressionContent.h>
```

Public Member Functions

- bool operator() (const T &) const
- template<typename T2 >
  bool operator() (const T2 &) const

0.15.192.1 Member Function Documentation

0.15.192.1.1 operator()() [1/2]

```
template<typename T >
bool smtrat::expression::ExpressionTypeChecker< T >::operator() (
            const T & ) const [inline]
```

0.15.192.1.2 operator()() [2/2]

```
template<typename T >
template<typename T2 >
bool smtrat::expression::ExpressionTypeChecker< T >::operator() (
            const T2 & ) const [inline]
```

0.15.193 smtrat::expression::ExpressionVisitor Class Reference

```
#include <ExpressionVisitor.h>
```

Public Types

- typedef std::function< void(const Expression &)> VisitorFunction

Public Member Functions

- void setPre (const VisitorFunction &f)
- void setPost (const VisitorFunction &f)
- void visit (const Expression &expression)
- void operator() (carl::Variable::Arg)
- void operator() (const ITEExpression &expr)
- void operator() (const QuantifierExpression &expr)
- void operator() (const UnaryExpression &expr)
- void operator() (const BinaryExpression &expr)
- void operator() (const NaryExpression &expr)

0.15.193.1 Member Typedef Documentation

0.15.193.1.1 VisitorFunction

```
typedef std::function<void(const Expression&)> smtrat::expression::ExpressionVisi
```

0.15.193.2 Member Function Documentation

0.15.193.2.1 operator()() [1/6]

```
void smtrat::expression::ExpressionVisitor::operator() (
            carl::Variable::Arg ) [inline]
```

0.15.193.2.2 operator()() [2/6]

```
void smtrat::expression::ExpressionVisitor::operator() (
            const BinaryExpression & expr ) [inline]
```

0.15.193.2.3 operator()() [3/6]

```
void smtrat::expression::ExpressionVisitor::operator() (
            const ITEExpression & expr ) [inline]
```

0.15.193.2.4 operator()() [4/6] void smtrat::expression::ExpressionVisitor::operator() (const NaryExpression & *expr*) [inline]

0.15.193.2.5 operator()() [5/6] void smtrat::expression::ExpressionVisitor::operator() (const QuantifierExpression & *expr*) [inline]

0.15.193.2.6 operator()() [6/6] void smtrat::expression::ExpressionVisitor::operator() (const UnaryExpression & *expr*) [inline]

0.15.193.2.7 setPost() void smtrat::expression::ExpressionVisitor::setPost (const VisitorFunction & *f*) [inline]

0.15.193.2.8 setPre() void smtrat::expression::ExpressionVisitor::setPre (const VisitorFunction & *f*) [inline]

0.15.193.2.9 visit() void smtrat::expression::ExpressionVisitor::visit (const Expression & *expression*) [inline]

0.15.194 smtrat::covering_ng::formula::node_ds::FALSE Struct Reference

`#include <FormulaEvaluation.h>`

0.15.195 smtrat::mcsat::FastParallelExplanation< Backends > Struct Template Reference

This explanation executes all given explanation in parallel processes and waits for the fastest explanation, returning the fastest delivered explanation, terminating all other parallel processes.

`#include <FastParallelExplanation.h>`

0.15.195.1 Detailed Description

template<typename... Backends>
struct smtrat::mcsat::FastParallelExplanation< Backends >

This explanation executes all given explanation in parallel processes and waits for the fastest explanation, returning the fastest delivered explanation, terminating all other parallel processes.

0.15.196 smtrat::mcsat::variableordering::detail::FeatureCollector< Objects > Struct Template Reference

This class manages features that are used to valuate variables on objects.

`#include <feature_based.h>`

Public Types

- using Extractor = std::function< double(Objects, carl::Variable)>
- using Valuation = std::vector< double >

Public Member Functions

- void addFeature (Extractor &&e, std::size_t level, double weight)
- Valuation valuateVariable (const Objects &o, carl::Variable v) const
- std::vector< carl::Variable > sortVariables (const Objects &o, std::vector< carl::Variable > vars) const

Data Fields

- std::vector< std::tuple< Extractor, std::size_t, double > > mFeatures

0.15.196.1 Detailed Description

template<typename Objects>
struct smtrat::mcsat::variableordering::detail::FeatureCollector< Objects >

This class manages features that are used to valuate variables on objects.
Each feature consists of a valuation function, a level and a weight. All feature valuations of a certain level are combined linearly using the respective weights. Valuations are then compared lexicographically.

0.15.196.2 Member Typedef Documentation

0.15.196.2.1 Extractor `template<typename Objects >`

```
using smtrat::mcsat::variableordering::detail::FeatureCollector< Objects >::Extractor = std↩
::function<double(Objects, carl::Variable)>
```

0.15.196.2.2 Valuation `template<typename Objects >`

```
using smtrat::mcsat::variableordering::detail::FeatureCollector< Objects >::Valuation = std↩
::vector<double>
```

0.15.196.3 Member Function Documentation

0.15.196.3.1 addFeature() `template<typename Objects >`

```
void smtrat::mcsat::variableordering::detail::FeatureCollector< Objects >::addFeature (
            Extractor && e,
            std::size_t level,
            double weight )  [inline]
```

0.15.196.3.2 sortVariables() `template<typename Objects >`

```
std::vector<carl::Variable> smtrat::mcsat::variableordering::detail::FeatureCollector< Objects
>::sortVariables (
            const Objects & o,
            std::vector< carl::Variable > vars ) const  [inline]
```

0.15.196.3.3 valuateVariable() `template<typename Objects >`

```
Valuation smtrat::mcsat::variableordering::detail::FeatureCollector< Objects >::valuate↩
Variable (
            const Objects & o,
            carl::Variable v ) const  [inline]
```

0.15.196.4 Field Documentation

0.15.196.4.1 mFeatures `template<typename Objects >`

```
std::vector<std::tuple<Extractor,std::size_t,double> > smtrat::mcsat::variableordering::detail::FeatureColle
Objects >::mFeatures
```

0.15.197 smtrat::parser::FixedWidthConstant< T > Struct Template Reference

Represents a constant of a fixed width.
`#include <TheoryTypes.h>`

Public Member Functions

- FixedWidthConstant ()
- FixedWidthConstant (const T &value, std::size_t width)
- bool operator< (const FixedWidthConstant &fwc) const

Data Fields

- T value
- std::size_t width

0.15.197.1 Detailed Description

template<typename T>
struct smtrat::parser::FixedWidthConstant< T >

Represents a constant of a fixed width.

0.15.197.2 Constructor & Destructor Documentation

0.15.197.2.1 FixedWidthConstant() [1/2] template<typename T >

```
smtrat::parser::FixedWidthConstant< T >::FixedWidthConstant ( ) [inline]
```

0.15.197.2.2 FixedWidthConstant() [2/2] template<typename T >

```
smtrat::parser::FixedWidthConstant< T >::FixedWidthConstant (
            const T & value,
            std::size_t width ) [inline]
```

0.15.197.3 Member Function Documentation

0.15.197.3.1 operator<() template<typename T >

```
bool smtrat::parser::FixedWidthConstant< T >::operator< (
            const FixedWidthConstant< T > & fwc ) const [inline]
```

0.15.197.4 Field Documentation

0.15.197.4.1 value template<typename T >

```
T smtrat::parser::FixedWidthConstant< T >::value
```

0.15.197.4.2 width template<typename T >

```
std::size_t smtrat::parser::FixedWidthConstant< T >::width
```

0.15.198 smtrat::parseformula::FormulaCollector Class Reference

`#include <parser_smtlib_utils.h>`

Public Types

- typedef types::AttributeValue Value

Public Member Functions

- void add (const smtrat::FormulaT &f)
- void addSoft (const FormulaT &f, Rational, const std::string &)
- void annotateName (const smtrat::FormulaT &, const std::string &)
- void check ()
- void declareFun (const carl::Variable &)
- void declareSort (const std::string &, const unsigned &)
- void defineSort (const std::string &, const std::vector< std::string > &, const carl::Sort &)
- void eliminateQuantifiers (const smtrat::qe::QEQuery &)
- void exit ()
- void getAssertions ()
- void getAllModels ()
- void getAssignment ()
- void getModel ()
- void getObjectives ()
- void getProof ()
- void getUnsatCore ()
- void getValue (const std::vector< carl::Variable > &)
- void addObjective (const smtrat::Poly &, smtrat::parser::OptimizationType)
- void pop (std::size_t)
- void push (std::size_t)
- void reset ()
- void resetAssertions ()
- void setLogic (const carl::Logic &l)
- int getExitCode () const
- FormulaT getFormula () const
- auto getLogic () const
- void addInstruction (std::function< void()> bind)
- bool hasInstructions () const
- void runInstructions ()
- void setArtificialVariables (std::vector< 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

0.15.198.1 Member Typedef Documentation

0.15.198.1.1 Value `typedef types::AttributeValue smtrat::parser::InstructionHandler::Value [inherited]`

0.15.198.2 Member Function Documentation

0.15.198.2.1 add()

```
void smtrat::parseformula::FormulaCollector::add (
    const smtrat::FormulaT & f ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

0.15.198.2.2 addInstruction()

```
void smtrat::parser::InstructionHandler::addInstruction (
    std::function< void() > bind ) [inline], [inherited]
```

0.15.198.2.3 addObjective()

```
void smtrat::parseformula::FormulaCollector::addObjective (
    const smtrat::Poly & ,
    smtrat::parser::OptimizationType ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

0.15.198.2.4 addSoft()

```
void smtrat::parseformula::FormulaCollector::addSoft (
    const FormulaT & f,
    Rational ,
    const std::string & ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

0.15.198.2.5 annotateName()

```
void smtrat::parseformula::FormulaCollector::annotateName (
    const smtrat::FormulaT & ,
    const std::string & ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

0.15.198.2.6 check()

```
void smtrat::parseformula::FormulaCollector::check ( ) [inline], [virtual]
```

Implements smtrat::parser::InstructionHandler.

0.15.198.2.7 cleanModel()

`void smtrat::parser::InstructionHandler::cleanModel ( smtrat::Model & model ) const [inline], [inherited]`

0.15.198.2.8 declareFun()

`void smtrat::parseformula::FormulaCollector::declareFun ( const carl::Variable & ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

0.15.198.2.9 declareSort()

`void smtrat::parseformula::FormulaCollector::declareSort ( const std::string & , const unsigned & ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

0.15.198.2.10 defineSort()

`void smtrat::parseformula::FormulaCollector::defineSort ( const std::string & , const std::vector< std::string > & , const carl::Sort & ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

0.15.198.2.11 diagnostic() [1/2]

`std::ostream& smtrat::parser::InstructionHandler::diagnostic ( ) [inline], [inherited]`

0.15.198.2.12 diagnostic() [2/2]

`void smtrat::parser::InstructionHandler::diagnostic ( const std::string & s ) [inline], [inherited]`

0.15.198.2.13 echo()

`virtual void smtrat::parser::InstructionHandler::echo ( const std::string & s ) [inline], [virtual], [inherited]`

0.15.198.2.14 eliminateQuantifiers()

`void smtrat::parseformula::FormulaCollector::eliminateQuantifiers ( const smtrat::qe::QEQuery & ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

0.15.198.2.15 error()

`OutputWrapper smtrat::parser::InstructionHandler::error ( ) [inline], [inherited]`

0.15.198.2.16 exit()

`void smtrat::parseformula::FormulaCollector::exit ( ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

0.15.198.2.17 get_info()

`const auto& smtrat::parser::InstructionHandler::get_info ( const std::string & key ) const [inline], [inherited]`

0.15.198.2.18 getAllModels()

`void smtrat::parseformula::FormulaCollector::getAllModels ( ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

0.15.198.2.19 getAssertions() `void smtrat::parseformula::FormulaCollector::getAssertions ( ) [inline], [virtual]`
Implements smtrat::parser::InstructionHandler.

0.15.198.2.20 getAssignment() `void smtrat::parseformula::FormulaCollector::getAssignment ( ) [inline], [virtual]`
Implements smtrat::parser::InstructionHandler.

0.15.198.2.21 getExitCode() `int smtrat::parseformula::FormulaCollector::getExitCode ( ) const [inline]`

0.15.198.2.22 getFormula() `FormulaT smtrat::parseformula::FormulaCollector::getFormula ( ) const [inline]`

0.15.198.2.23 getInfo() `void smtrat::parser::InstructionHandler::getInfo ( const std::string & key ) [inline], [inherited]`

0.15.198.2.24 getLogic() `auto smtrat::parseformula::FormulaCollector::getLogic ( ) const [inline]`

0.15.198.2.25 getModel() `void smtrat::parseformula::FormulaCollector::getModel ( ) [inline], [virtual]`
Implements smtrat::parser::InstructionHandler.

0.15.198.2.26 getObjectives() `void smtrat::parseformula::FormulaCollector::getObjectives ( ) [inline], [virtual]`
Implements smtrat::parser::InstructionHandler.

0.15.198.2.27 getOption() `void smtrat::parser::InstructionHandler::getOption ( const std::string & key ) [inline], [inherited]`

0.15.198.2.28 getProof() `void smtrat::parseformula::FormulaCollector::getProof ( ) [inline], [virtual]`
Implements smtrat::parser::InstructionHandler.

0.15.198.2.29 getUnsatCore() `void smtrat::parseformula::FormulaCollector::getUnsatCore ( ) [inline], [virtual]`
Implements smtrat::parser::InstructionHandler.

0.15.198.2.30 getValue() `void smtrat::parseformula::FormulaCollector::getValue ( const std::vector< carl::Variable > & ) [inline], [virtual]`
Implements smtrat::parser::InstructionHandler.

0.15.198.2.31 has_info()

`bool smtrat::parser::InstructionHandler::has_info ( const std::string & key ) const [inline], [inherited]`

0.15.198.2.32 hasInstructions()

`bool smtrat::parser::InstructionHandler::hasInstructions ( ) const [inline], [inherited]`

0.15.198.2.33 info()

`OutputWrapper smtrat::parser::InstructionHandler::info ( ) [inline], [inherited]`

0.15.198.2.34 option()

`template<typename T > T smtrat::parser::InstructionHandler::option ( const std::string & key ) const [inline], [inherited]`

0.15.198.2.35 pop()

`void smtrat::parseformula::FormulaCollector::pop ( std::size_t ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

0.15.198.2.36 printInstruction()

`bool smtrat::parser::InstructionHandler::printInstruction ( ) const [inline], [inherited]`

0.15.198.2.37 push()

`void smtrat::parseformula::FormulaCollector::push ( std::size_t ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

0.15.198.2.38 regular() [1/2]

`std::ostream& smtrat::parser::InstructionHandler::regular ( ) [inline], [inherited]`

0.15.198.2.39 regular() [2/2]

`void smtrat::parser::InstructionHandler::regular ( const std::string & s ) [inline], [inherited]`

0.15.198.2.40 reset()

`void smtrat::parseformula::FormulaCollector::reset ( ) [inline], [virtual]`

Reimplemented from smtrat::parser::InstructionHandler.

0.15.198.2.41 resetAssertions()

`void smtrat::parseformula::FormulaCollector::resetAssertions ( ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

0.15.198.2.42 runInstructions()

`void smtrat::parser::InstructionHandler::runInstructions ( ) [inline], [inherited]`

0.15.198.2.43 setArtificialVariables()

`void smtrat::parser::InstructionHandler::setArtificial↩ Variables ( std::vector< smtrat::ModelVariable > && vars ) [inline], [inherited]`

0.15.198.2.44 setInfo()

`void smtrat::parser::InstructionHandler::setInfo ( const Attribute & attr ) [inline], [inherited]`

0.15.198.2.45 setLogic()

`void smtrat::parseformula::FormulaCollector::setLogic ( const carl::Logic & l ) [inline], [virtual]`

Implements smtrat::parser::InstructionHandler.

0.15.198.2.46 setOption()

`void smtrat::parser::InstructionHandler::setOption ( const Attribute & option ) [inline], [inherited]`

0.15.198.2.47 setStream()

`void smtrat::parser::InstructionHandler::setStream ( const std::string & s, std::ostream & os ) [inline], [protected], [inherited]`

0.15.198.2.48 warn()

`OutputWrapper smtrat::parser::InstructionHandler::warn ( ) [inline], [inherited]`

0.15.198.3 Field Documentation

0.15.198.3.1 infos

`VariantMap<std::string, Value> smtrat::parser::InstructionHandler::infos [protected], [inherited]`

0.15.198.3.2 mDiagnostic

`std::ostream smtrat::parser::InstructionHandler::mDiagnostic [protected], [inherited]`

0.15.198.3.3 mRegular

`std::ostream smtrat::parser::InstructionHandler::mRegular [protected], [inherited]`

0.15.198.3.4 options

`VariantMap<std::string, Value> smtrat::parser::InstructionHandler::options [protected], [inherited]`

0.15.198.3.5 streams

`std::map<std::string, std::ofstream> smtrat::parser::InstructionHandler↩ ::streams [protected], [inherited]`

0.15.199 smtrat::ICPModule< Settings >::formulaPtrComp Struct Reference

`#include <ICPModule.h>`

Public Member Functions

- bool operator() (const FormulaT &_lhs, const FormulaT &_rhs) const

0.15.199.1 Member Function Documentation

0.15.199.1.1 operator()() template<class Settings >

```
bool smtrat::ICPModule< Settings >::formulaPtrComp::operator() (
            const FormulaT & _lhs,
            const FormulaT & _rhs ) const  [inline]
```

0.15.200 smtrat::FormulaWithOrigins Class Reference

Stores a formula along with its origins.

`#include <ModuleInput.h>`

Public Member Functions

- FormulaWithOrigins ()
- FormulaWithOrigins (const FormulaT &_formula)

  Constructs a formula with empty origins.
- FormulaWithOrigins (const FormulaT &_formula, const std::shared_ptr< FormulasT > &_origins)

  Constructs a formula with the given origins.
- FormulaWithOrigins (const FormulaWithOrigins &)
- const FormulaT & formula () const
- bool hasOrigins () const
- const FormulasT & origins () const
- void setDeducted (bool _deducted) const

  Sets the deduction flag to the given value.
- bool deducted () const

Friends

- class ModuleInput
- bool operator< (const FormulaWithOrigins &_fwoA, const FormulaWithOrigins &_fwoB)
- bool operator== (const FormulaWithOrigins &_fwoA, const FormulaWithOrigins &_fwoB)

0.15.200.1 Detailed Description

Stores a formula along with its origins.

0.15.200.2 Constructor & Destructor Documentation

0.15.200.2.1 FormulaWithOrigins() [1/4] smtrat::FormulaWithOrigins::FormulaWithOrigins ( )

0.15.200.2.2 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.

0.15.200.2.3 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.

0.15.200.2.4 FormulaWithOrigins() [4/4] `smtrat::FormulaWithOrigins::FormulaWithOrigins ( const FormulaWithOrigins & )`

0.15.200.3 Member Function Documentation

0.15.200.3.1 deducted() `bool smtrat::FormulaWithOrigins::deducted ( ) const [inline]`

Returns

The deduction flag, which indicates, that this formula g is a direct sub-formula of a conjunction of formulas (and g f_1 .. f_n), and, that (implies (and f_1 .. f_n) g) holds.

0.15.200.3.2 formula() `const FormulaT& smtrat::FormulaWithOrigins::formula ( ) const [inline]`

Returns

A constant reference to the formula.

0.15.200.3.3 hasOrigins() `bool smtrat::FormulaWithOrigins::hasOrigins ( ) const [inline]`

Returns

true, if this sub-formula of the module input has origins.

0.15.200.3.4 origins() `const FormulasT& smtrat::FormulaWithOrigins::origins ( ) const [inline]`

Returns

A constant reference to the origins.

0.15.200.3.5 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.

0.15.200.4 Friends And Related Function Documentation

0.15.200.4.1 ModuleInput `friend class ModuleInput [friend]`

0.15.200.4.2 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.

0.15.200.4.3 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.

0.15.201 smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices > Class Template Reference

This class implements a forward hypergraph.
`#include <ForwardHyperGraph.h>`

Data Structures

- struct Edge

  Internal type of an edge.

Public Types

- using Vertex = std::size_t

  Internal type of a vertex.

Public Member Functions

- Vertex newVertex (const VertexProperty &vp)

  Returns a new vertex that is labeled with the given vertex property.
- Edge & newEdge (Vertex source, const EdgeProperty &ep)

  Creates a new edge where the given vertex is the incoming vertex that is labeled with the given edge property.
- Edge & newEdge (const VertexProperty &source, const EdgeProperty &ep)

Creates a new edge where the given vertex is the incoming vertex that is labeled with the given edge property.

- const VertexProperty & operator[] (Vertex v) const

  Retrieves the vertex property for the given vertex.
- VertexProperty & operator[] (Vertex v)

  Retrieves the vertex property for the given vertex.
- const std::vector< Edge > & out (Vertex v) const

  Retrieves all outgoing edges for the given vertex.
- const std::vector< Edge > & out (const VertexProperty &v) const

  Retrieves all outgoing edges for the given vertex.
- Vertex begin () const

  Returns the first vertex.
- Vertex end () const

  Returns the past-the-last vertex.
- std::size_t size () const

  Returns the number of vertices.
- void toDot (const std::string &filename) const

  Writes a representation of this graph to the given file.

Friends

- template<typename VP , typename EP >
  std::ostream & operator<< (std::ostream &os, const ForwardHyperGraph< VP, EP > &fhg)

  Writes a textual representation of this graph to the given stream.

0.15.201.1 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.

0.15.201.2 Member Typedef Documentation

0.15.201.2.1 Vertex

```
template<typename VertexProperty , typename EdgeProperty , bool Unique↩
Vertices = true>
using smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Vertex =
std::size_t
```

Internal type of a vertex.

0.15.201.3 Member Function Documentation

0.15.201.3.1 begin()

```
template<typename VertexProperty , typename EdgeProperty , bool Unique↩
Vertices = true>
Vertex smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::begin ( )
const [inline]
```

Returns the first vertex.

0.15.201.3.2 end()

```
template<typename VertexProperty , typename EdgeProperty , bool Unique↩
Vertices = true>
Vertex smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::end ( )
const [inline]
```

Returns the past-the-last vertex.

0.15.201.3.3 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.

0.15.201.3.4 newEdge() [2/2]

```
template<typename VertexProperty , typename EdgeProperty , bool
UniqueVertices = true>
Edge& smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::newEdge (
            Vertex source,
            const EdgeProperty & ep ) [inline]
```

Creates a new edge where the given vertex is the incoming vertex that is labeled with the given edge property.

0.15.201.3.5 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.

0.15.201.3.6 operator[]() [1/2]

```
template<typename VertexProperty , typename EdgeProperty , bool
UniqueVertices = true>
VertexProperty& smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >↩
::operator[] (
            Vertex v ) [inline]
```

Retrieves the vertex property for the given vertex.
This method is disabled if UniqueVertices is set to true.

0.15.201.3.7 operator[]() [2/2]

```
template<typename VertexProperty , typename EdgeProperty , bool
UniqueVertices = true>
const VertexProperty& smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices
>::operator[] (
            Vertex v ) const [inline]
```

Retrieves the vertex property for the given vertex.

0.15.201.3.8 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.

0.15.201.3.9 out() [2/2]

```
template<typename VertexProperty , typename EdgeProperty , bool Unique↩
Vertices = true>
const std::vector<Edge>& smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, Unique↩
Vertices >::out (
            Vertex v ) const  [inline]
```

Retrieves all outgoing edges for the given vertex.

0.15.201.3.10 size()

```
template<typename VertexProperty , typename EdgeProperty , bool Unique↩
Vertices = true>
std::size_t smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::size (
) const  [inline]
```

Returns the number of vertices.

0.15.201.3.11 toDot()

```
template<typename VertexProperty , typename EdgeProperty , bool Unique↩
Vertices = true>
void smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::toDot (
            const std::string & filename ) const
```

Writes a representation of this graph to the given file.

0.15.201.4 Friends And Related Function Documentation

0.15.201.4.1 operator<<

```
template<typename VertexProperty , typename EdgeProperty , bool Unique↩
Vertices = true>
template<typename VP , typename EP >
std::ostream& operator<< (
            std::ostream & os,
            const ForwardHyperGraph< VP, EP > & fhg )  [friend]
```

Writes a textual representation of this graph to the given stream.

0.15.202 smtrat::FouMoModule< Settings > Class Template Reference

A module which applies the Fourier-Motzkin algorithm.

```
#include <FouMoModule.h>
```

Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- 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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.
- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)
- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

Copies the infeasible subsets of the passed formula.

- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.202.1 Detailed Description

template<typename Settings>
class smtrat::FouMoModule< Settings >

A module which applies the Fourier-Motzkin algorithm.

0.15.202.2 Member Typedef Documentation

0.15.202.2.1 SettingsType `template<typename Settings >`

`typedef Settings smtrat::FouMoModule< Settings >::SettingsType`

0.15.202.3 Member Enumeration Documentation

0.15.202.3.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

0.15.202.4 Constructor & Destructor Documentation

0.15.202.4.1 FouMoModule() `template<typename Settings >`

```
smtrat::FouMoModule< Settings >::FouMoModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = NULL )
```

0.15.202.4.2 ~FouMoModule() `template<typename Settings >`

`smtrat::FouMoModule< Settings >::~FouMoModule ( ) [inline]`

0.15.202.5 Member Function Documentation

0.15.202.5.1 add() `bool smtrat::Module::add (`

`ModuleInput::const_iterator _subformula ) [inherited]`

The module has to take the given sub-formula of the received formula into account.

Parameters

_subformula	The sub-formula to take additionally into account.

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

0.15.202.5.2 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 >.

0.15.202.5.3 addCore() [1/2]

```
virtual bool smtrat::Module::addCore (
    [[maybe_unused] ] 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.

0.15.202.5.4 addCore() [2/2]

```
template<typename Settings >
bool smtrat::FouMoModule< Settings >::addCore (
    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.

0.15.202.5.5 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula
(
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.202.5.6 addLemma()

```
void smtrat::Module::addLemma (
    const FormulaT & _lemma,
    const LemmaType & _lt = LemmaType::NORMAL,
    const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline],
[inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.202.5.7 addOrigin()

```
void smtrat::Module::addOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.202.5.8 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.202.5.9 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.202.5.10 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.202.5.11 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.

0.15.202.5.12 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns

All satisfying assignments, if existent.

0.15.202.5.13 anAnswerFound()

```
bool smtrat::Module::anAnswerFound ( ) const [inline], [protected],
[inherited]
```

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.202.5.14 answerFound()

```
const smtrat::Conditionals& smtrat::Module::answerFound ( ) const
[inline], [inherited]
```

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.202.5.15 backendsModel()

```
const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]
```

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.202.5.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.202.5.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.202.5.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.202.5.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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.

Parameters

_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

0.15.202.5.20 check()

```
Answer smtrat::Module::check (
    bool _final = false,
    bool _full = true,
    carl::Variable _objective = carl::Variable::NO_VARIABLE ) [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.202.5.21 checkCore()

```
template<typename 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.

0.15.202.5.22 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.

0.15.202.5.23 checkModel() `unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]`

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.202.5.24 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.202.5.25 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

0.15.202.5.26 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

0.15.202.5.27 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

0.15.202.5.28 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.202.5.29 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

0.15.202.5.30 collectOrigins() [2/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.
If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.202.5.31 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

0.15.202.5.32 constraintsToInform() `const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns

The constraints which the backends must be informed about.

0.15.202.5.33 currentlySatisfied() `virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.202.5.34 currentlySatisfiedByBackend() `unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.202.5.35 deinform() `void smtrat::Module::deinform ( const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.202.5.36 deinformCore() `virtual void smtrat::Module::deinformCore ( [[maybe_unused] ] 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.

0.15.202.5.37 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

0.15.202.5.38 eraseSubformulaFromPassedFormula()

```
ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (
            ModuleInput::iterator _subformula,
            bool _ignoreOrigins = false ) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_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 >.

0.15.202.5.39 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

0.15.202.5.40 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
            const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

0.15.202.5.41 firstSubformulaToPass()

```
ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]
```

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.202.5.42 firstUncheckedReceivedSubformula() `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.202.5.43 freeSplittingVariable() `static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.202.5.44 generateTrivialInfeasibleSubset() `void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.202.5.45 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.202.5.46 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.202.5.47 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.202.5.48 getInfeasibleSubsets() [2/2] `std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets ( const Module & _backend ) const [protected], [inherited]`

Get the infeasible subsets the given backend provides.
Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

Parameters

_backend	The backend from which to obtain the infeasible subsets.

Returns

The infeasible subsets the given backend provides.

0.15.202.5.49 getInformationRelevantFormulas() `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.202.5.50 getModelEqualities() `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.202.5.51 getOrigins() [1/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.202.5.52 getOrigins() [2/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.202.5.53 getOrigins() [3/3] `const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]`

Gets the origins of the passed formula at the given position.

Parameters

_formula	The position of a formula in the passed formulas.

Returns

The origins of the passed formula at the given position.

0.15.202.5.54 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.

0.15.202.5.55 hasLemmas() `bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

0.15.202.5.56 hasValidInfeasibleSubset() `bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]`

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.202.5.57 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.202.5.58 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasible↩ Subsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.202.5.59 inform() `bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.
It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

Parameters

_constraint	The constraint to inform about.

Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

0.15.202.5.60 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.

0.15.202.5.61 informCore() `virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.202.5.62 informedConstraints() `const carl::FastSet<FormulaT>& smtrat::Module::informedConstraints ( ) const [inline], [inherited]`

Returns

The position of the first constraint of which no backend has been informed about.

0.15.202.5.63 init() `void smtrat::Module::init ( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not be called twice and only before the first runBackends call.
Reimplemented in smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntBlastModule< Settings >, smtrat::FPPModule< Settings >, smtrat::CurryModule< Settings >, and smtrat::BVModule< Settings >.

0.15.202.5.64 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.202.5.65 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.

0.15.202.5.66 isPreprocessor() `bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.202.5.67 lemmas() `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.202.5.68 merge()

```
std::vector< FormulaT > smtrat::Module::merge (
    const std::vector< FormulaT > & _vecSetA,
    const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]
```

Merges the two vectors of sets into the first one.
({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

Parameters

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns

The vector being the two given vectors merged.

0.15.202.5.69 model()

```
const Model& smtrat::Module::model ( ) const [inline], [inherited]
```

Returns

The assignment of the current satisfiable result, if existent.

0.15.202.5.70 modelsDisjoint()

```
bool smtrat::Module::modelsDisjoint (
    const Model & _modelA,
    const Model & _modelB ) [static], [protected], [inherited]
```

Checks whether there is no variable assigned by both models.

Parameters

_modelA	The first model to check for.
_modelB	The second model to check for.

Returns

true, if there is no variable assigned by both models; false, otherwise.

0.15.202.5.71 moduleName()

```
template<typename Settings >
std::string smtrat::FouMoModule< Settings >::moduleName ( ) const [inline], [virtual]
```

Returns

The name of the given module type as name.

Reimplemented from smtrat::Module.

0.15.202.5.72 objective()

```
carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]
```

0.15.202.5.73 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.202.5.74 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.202.5.75 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.202.5.76 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.202.5.77 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.202.5.78 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.202.5.79 printAllModels()

`void smtrat::Module::printAllModels ( std::ostream & _out = std::cout ) [inherited]`

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.202.5.80 printInfeasibleSubsets()

`void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.202.5.81 printModel()

`void smtrat::Module::printModel ( std::ostream & _out = std::cout ) const [inherited]`

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.202.5.82 printPassedFormula()

`void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.202.5.83 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.202.5.84 probablyLooping()

`bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.202.5.85 receivedFormulaChecked() `void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.202.5.86 receivedFormulasAsInfeasibleSubset() `void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

0.15.202.5.87 receivedVariable() `bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

0.15.202.5.88 remove() `void smtrat::Module::remove ( ModuleInput::const_iterator _subformula ) [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.
However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

Parameters

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

0.15.202.5.89 removeCore() [1/2] `virtual void smtrat::Module::removeCore ( [[maybe_unused] ] 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.

0.15.202.5.90 removeCore() [2/2] `template<typename Settings > void smtrat::FouMoModule< Settings >::removeCore ( ModuleInput::const_iterator _subformula )`

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.

0.15.202.5.91 removeOrigin() `std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩Origin (`

```
ModuleInput::iterator _formula,
const FormulaT & _origin )  [inline], [protected], [inherited]
```

0.15.202.5.92 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins (
ModuleInput::iterator _formula,
const std::shared_ptr< std::vector< FormulaT >> & _origins )  [inline], [protected], [inherited]
```

0.15.202.5.93 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.202.5.94 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]
```

Returns

A reference to the formula passed to this module.

0.15.202.5.95 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( )  [inline], [protected], [virtual], [inherited]
```

Reimplemented in smtrat::PModule.

0.15.202.5.96 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.

0.15.202.5.97 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.

0.15.202.5.98 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.

0.15.202.5.99 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.

0.15.202.5.100 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 !=.

0.15.202.5.101 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.202.5.102 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 >.

0.15.202.5.103 updateLemmas()

`void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.202.5.104 updateModel() `template<typename Settings > void smtrat::FouMoModule< Settings >::updateModel ( ) const [virtual]`

Updates the current assignment into the model.
Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.
Reimplemented from smtrat::Module.

0.15.202.5.105 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).

0.15.202.6 Field Documentation

0.15.202.6.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.202.6.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.202.6.3 mBackendsFoundAnswer `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.202.6.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.202.6.5 mFinalCheck `bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.202.6.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.202.6.7 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.

0.15.202.6.8 mFirstUncheckedReceivedSubformula `ModuleInput::const_iterator smtrat::Module↩ ::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.202.6.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.202.6.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.202.6.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.202.6.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.202.6.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std↩::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.202.6.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.202.6.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.202.6.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.202.6.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.202.6.18 mObjectiveVariable

`carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.202.6.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplitting↩Variables [static], [inherited]`

Reusable splitting variables.

0.15.202.6.20 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.202.6.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.202.6.22 mSolverState `std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.202.6.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

0.15.202.6.24 mUsedBackends `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.202.6.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.203 smtrat::qe::fm::FourierMotzkinQE Class Reference

Provides a simple implementation for Fourier Motzkin variable elimination for linear, existentially quantified constraints.

`#include <FourierMotzkinQE.h>`

Public Types

- using FormulaPartition = std::vector< std::vector< FormulaT > >

Public Member Functions

- FourierMotzkinQE (const FormulaT &qfree, const QEQuery &quantifiers)
- FormulaT eliminateQuantifiers ()

0.15.203.1 Detailed Description

Provides a simple implementation for Fourier Motzkin variable elimination for linear, existentially quantified constraints.

0.15.203.2 Member Typedef Documentation

0.15.203.2.1 FormulaPartition `using smtrat::qe::fm::FourierMotzkinQE::FormulaPartition = std↩ ::vector<std::vector<FormulaT> >`

0.15.203.3 Constructor & Destructor Documentation

0.15.203.3.1 FourierMotzkinQE()

```
smtrat::qe::fm::FourierMotzkinQE::FourierMotzkinQE (
    const FormulaT & qfree,
    const QEQuery & quantifiers ) [inline]
```

0.15.203.4 Member Function Documentation

0.15.203.4.1 eliminateQuantifiers() `FormulaT smtrat::qe::fm::FourierMotzkinQE::eliminateQuantifiers ( )`

0.15.204 smtrat::FPPModule< Settings > Class Template Reference

`#include <FPPModule.h>`

Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- FPPModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=NULL)
- ~FPPModule ()
- bool informCore (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void init ()

  Informs all backends about the so far encountered constraints, which have not yet been communicated.
- bool addCore (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- void removeCore (ModuleInput::const_iterator _subformula)

  Removes the subformula of the received formula at the given position to the considered ones of this module.
- void updateModel () const

  Updates the current assignment into the model.
- Answer checkCore ()

  Checks the received formula for consistency.
- bool isPreprocessor () const
- bool appliedPreprocessing () const
- bool add (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- void remove (ModuleInput::const_iterator _subformula)

  Removes everything related to the given sub-formula of the received formula.
- Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)

  Checks the received formula for consistency.
- Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- std::pair< bool, FormulaT > getReceivedFormulaSimplified ()
- bool inform (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void deinform (const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual void updateAllModels ()

  Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
- virtual std::list< std::vector< carl::Variable > > getModelEqualities () const

  Partition the variables from the current model into equivalence classes according to their assigned value.
- unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const
- virtual unsigned currentlySatisfied (const FormulaT &) const

- bool receivedVariable (carl::Variable::Arg _var) const
- Answer solverState () const
- std::size_t id () const
- void setId (std::size_t _id)

  Sets this 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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] 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 _useReceivedFormulaAs↩ Premise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &↩ _premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.204.1 Member Typedef Documentation

0.15.204.1.1 SettingsType

```
template<typename Settings >
typedef Settings smtrat::FPPModule< Settings >::SettingsType
```

0.15.204.2 Member Enumeration Documentation

0.15.204.2.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

0.15.204.3 Constructor & Destructor Documentation

0.15.204.3.1 FPPModule()

```
template<typename Settings >
smtrat::FPPModule< Settings >::FPPModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = NULL )
```

0.15.204.3.2 ~FPPModule()

```
template<typename Settings >
smtrat::FPPModule< Settings >::~FPPModule ( )
```

0.15.204.4 Member Function Documentation

0.15.204.4.1 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.

0.15.204.4.2 addConstraintToInform()

```
void smtrat::Module::addConstraintToInform (
    const FormulaT & _constraint ) [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

0.15.204.4.3 addCore() [1/2]

```
virtual bool smtrat::Module::addCore (
    [[maybe_unused] ] 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.

0.15.204.4.4 addCore() [2/2]

```
template<typename Settings >
bool smtrat::FPPModule< Settings >::addCore (
    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.

0.15.204.4.5 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula
(
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.204.4.6 addLemma()

```
void smtrat::Module::addLemma (
    const FormulaT & _lemma,
    const LemmaType & _lt = LemmaType::NORMAL,
    const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline], [inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.204.4.7 addOrigin()

```
void smtrat::Module::addOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.204.4.8 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.204.4.9 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.204.4.10 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula,
            const FormulaT & _origin )  [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.204.4.11 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.

0.15.204.4.12 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const  [inline],
[inherited]
```

Returns

All satisfying assignments, if existent.

0.15.204.4.13 anAnswerFound() `bool smtrat::Module::anAnswerFound ( ) const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.204.4.14 answerFound() `const smtrat::Conditionals& smtrat::Module::answerFound ( ) const [inline], [inherited]`

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.204.4.15 appliedPreprocessing() `bool smtrat::PModule::appliedPreprocessing ( ) const [inline], [inherited]`

Returns

true, if the current received formula has been simplified and the result of this simplification is stored in the passed formula.

0.15.204.4.16 backendsModel() `const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.204.4.17 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.204.4.18 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.204.4.19 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
```

```
const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline],
[protected], [inherited]
```

0.15.204.4.20 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.204.4.21 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.

0.15.204.4.22 checkCore()

```
template<typename Settings >
Answer smtrat::FPPModule< Settings >::checkCore ( ) [virtual]
```

Checks the received formula for consistency.

Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.204.4.23 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.

0.15.204.4.24 checkModel()

```
unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]
```

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.204.4.25 cleanModel()

```
void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]
```

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.204.4.26 clearLemmas()

```
void smtrat::Module::clearLemmas ( ) [inline], [inherited]
```

Deletes all yet found lemmas.

0.15.204.4.27 clearModel()

```
void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]
```

Clears the assignment, if any was found.

0.15.204.4.28 clearModels()

```
void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]
```

Clears all assignments, if any was found.

0.15.204.4.29 clearPassedFormula()

```
void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]
```

0.15.204.4.30 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

0.15.204.4.31 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.

0.15.204.4.32 collectTheoryPropagations()

```
void smtrat::Module::collectTheoryPropagations ( ) [inherited]
```

0.15.204.4.33 constraintsToInform()

```
const carl::FastSet<FormulaT>& smtrat::Module::constraints↩
ToInform ( ) const [inline], [inherited]
```

Returns

The constraints which the backends must be informed about.

0.15.204.4.34 currentlySatisfied()

```
virtual unsigned smtrat::Module::currentlySatisfied (
    const FormulaT & ) const [inline], [virtual], [inherited]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.204.4.35 currentlySatisfiedByBackend()

```
unsigned smtrat::Module::currentlySatisfiedByBackend (
    const FormulaT & _formula ) const [inherited]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.204.4.36 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.

0.15.204.4.37 deinformCore()

`virtual void smtrat::Module::deinformCore ( [[maybe_unused] ] 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.

0.15.204.4.38 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

0.15.204.4.39 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 >.

0.15.204.4.40 excludeNotReceivedVariablesFromModel()

`void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

0.15.204.4.41 findBestOrigin()

`std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin ( const std::vector< FormulaT > & _origins ) const [protected], [inherited]`

Parameters

_origins	

Returns

0.15.204.4.42 firstSubformulaToPass()

`ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.204.4.43 firstUncheckedReceivedSubformula()

`ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.204.4.44 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.204.4.45 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.204.4.46 getBackendsAllModels()

`void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.204.4.47 getBackendsModel()

`void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.204.4.48 getInfeasibleSubsets() [1/2]

`void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.204.4.49 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.

0.15.204.4.50 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.204.4.51 getModelEqualities()

`std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.204.4.52 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.204.4.53 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.204.4.54 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.

0.15.204.4.55 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.

0.15.204.4.56 hasLemmas() `bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

0.15.204.4.57 hasValidInfeasibleSubset() `bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]`

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.204.4.58 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.204.4.59 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.204.4.60 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.

0.15.204.4.61 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.

0.15.204.4.62 informCore() [1/2]

`virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.204.4.63 informCore() [2/2]

`template<typename Settings > bool smtrat::FPPModule< Settings >::informCore ( 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.

0.15.204.4.64 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.

0.15.204.4.65 init() `template<typename Settings >`
`void smtrat::FPPModule< Settings >::init ( ) [virtual]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not and will not be called more than once and only before the first runBackends call.
Reimplemented from smtrat::Module.

0.15.204.4.66 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.204.4.67 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.

0.15.204.4.68 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().

0.15.204.4.69 lemmas() `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.204.4.70 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.

0.15.204.4.71 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.204.4.72 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.

0.15.204.4.73 moduleName()

```
virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]
```

Returns

The name of the given module type as name.

Reimplemented in smtrat::VSModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::STropModule< Settings >, smtrat::SplitSOSModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntEqModule< Settings >, smtrat::IntBlastModule< Settings >, smtrat::IncWidthModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::GBModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::CurryModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::BVModule< Settings >, and smtrat::BEModule< Settings >.

0.15.204.4.74 objective()

```
carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]
```

0.15.204.4.75 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
    const FormulaT & _origin ) const [protected], [inherited]
```

0.15.204.4.76 passedFormulaBegin()

```
ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]
```

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.204.4.77 passedFormulaEnd()

```
ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]
```

Returns

An iterator to the end of the passed formula.

0.15.204.4.78 pPassedFormula() `const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.204.4.79 pReceivedFormula() `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.204.4.80 print() `void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.204.4.81 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.

0.15.204.4.82 printInfeasibleSubsets() `void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.204.4.83 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.

0.15.204.4.84 printPassedFormula()

`void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.204.4.85 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.204.4.86 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.

0.15.204.4.87 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.204.4.88 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

0.15.204.4.89 receivedVariable()

`bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

0.15.204.4.90 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.

0.15.204.4.91 removeCore() [1/2]

```
virtual void smtrat::Module::removeCore (
            [[maybe_unused] ] 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.

0.15.204.4.92 removeCore() [2/2]

```
template<typename Settings >
void smtrat::FPPModule< Settings >::removeCore (
            ModuleInput::const_iterator _subformula )
```

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.

0.15.204.4.93 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origin (
            ModuleInput::iterator _formula,
            const FormulaT & _origin ) [inline], [protected], [inherited]
```

0.15.204.4.94 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

0.15.204.4.95 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.204.4.96 rReceivedFormula() `const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]`

Returns

A reference to the formula passed to this module.

0.15.204.4.97 runBackends() [1/2] `virtual Answer smtrat::PModule::runBackends ( ) [inline], [virtual], [inherited]`

Reimplemented from smtrat::Module.

0.15.204.4.98 runBackends() [2/2]

```
Answer smtrat::PModule::runBackends (
    bool _final,
    bool _full,
    carl::Variable _objective ) [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

Parameters

_final	true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT.
_full	false, if this module should avoid too expensive procedures and rather return unknown instead.
_objective	if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good.

Returns

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.204.4.99 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.

0.15.204.4.100 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.

0.15.204.4.101 solverState() `Answer smtrat::Module::solverState ( ) const [inline], [inherited]`

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.204.4.102 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 !=.

0.15.204.4.103 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.204.4.104 updateAllModels()

`void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.204.4.105 updateLemmas()

`void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.204.4.106 updateModel()

`template<typename 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.

0.15.204.4.107 usedBackends()

`const std::vector<Module*>& smtrat::Module::usedBackends ( ) const [inline], [inherited]`

Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.204.5 Field Documentation

0.15.204.5.1 mAllBackends

`std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.204.5.2 mAllModels

`std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.204.5.3 mBackendsFoundAnswer

`std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.204.5.4 mConstraintsToInform

`carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.204.5.5 mFinalCheck

`bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.204.5.6 mFirstPosInLastBranches

`std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.204.5.7 mFirstSubformulaToPass

`ModuleInput::iterator smtrat::Module::mFirstSubformulaToPass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.204.5.8 mFirstUncheckedReceivedSubformula

`ModuleInput::const_iterator smtrat::Module::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.204.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.204.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.204.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.204.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.204.5.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.204.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.204.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.204.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.204.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.204.5.18 mObjectiveVariable

`carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.204.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]`

Reusable splitting variables.

0.15.204.5.20 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.204.5.21 mSmallerMusesCheckCounter

`unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.204.5.22 mSolverState

`std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.204.5.23 mTheoryPropagations

`std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]`

0.15.204.5.24 mUsedBackends `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.204.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.205 smtrat::mcsat::FullParallelExplanation< Backends > Struct Template Reference

This explanation executes all given explanation parallel in multiple threads and waits until every single one has finished, returning the result of the first listed explanation.

`#include <FullParallelExplanation.h>`

0.15.205.1 Detailed Description

template<typename... Backends>
struct smtrat::mcsat::FullParallelExplanation< Backends >

This explanation executes all given explanation parallel in multiple threads and waits until every single one has finished, returning the result of the first listed explanation.

0.15.206 smtrat::cad::FullSampleComparator< Iterator, Strategy > Struct Template Reference

`#include <SampleComparator.h>`

0.15.207 smtrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::T > Struct Template Reference

`#include <SampleComparator.h>`

0.15.208 smtrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::Type > Struct Template Reference

`#include <SampleComparator.h>`

0.15.209 smtrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::Value > Struct Template Reference

`#include <SampleComparator.h>`

0.15.210 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

0.15.210.1 Constructor & Destructor Documentation

0.15.210.1.1 ~FunctionInstantiator()

```
virtual smtrat::parser::FunctionInstantiator::~Function↩
Instantiator ( ) [inline], [virtual]
```

0.15.210.2 Member Function Documentation

0.15.210.2.1 convert() [1/2]

```
template<typename T >
bool smtrat::parser::FunctionInstantiator::convert (
            const std::vector< types::TermType > & from,
            std::vector< T > & to ) const [inline]
```

0.15.210.2.2 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]
```

0.15.210.2.3 operator()()

```
virtual bool smtrat::parser::FunctionInstantiator::operator() (
            const std::vector< types::TermType > & ,
            types::TermType & ,
            TheoryError & errors ) const [inline], [virtual]
```

0.15.211 smtrat::GBModule< Settings > Class Template Reference

A solver module based on Groebner basis.

```
#include <GBModule.h>
```

Public Types

- typedef Settings SettingsType
- typedef Settings::Order Order
- typedef Settings::PolynomialWithReasons GBPolynomial
- typedef Settings::MultivariateIdeal Ideal
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- GBModule (const ModuleInput *const, Conditionals &, Manager *const =NULL)
- virtual ~GBModule ()
- bool addCore (ModuleInput::const_iterator _formula)
- virtual Answer checkCore ()

  Checks the received formula for consistency.
- void removeCore (ModuleInput::const_iterator _formula)
- 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)
- void popBacktrackPoint (ModuleInput::const_iterator btpoint)
- bool saveState ()
- FormulasT generateReasons (const carl::BitVector &reasons)
- FormulaSetT generateReasonSet (const carl::BitVector &reasons)
- void passGB ()
- void knownConstraintDeduction (const std::list< std::pair< carl::BitVector, carl::BitVector > > &deductions)

- void newConstraintDeduction ()
- void factorisedConstraintDeduction (const std::list< GBPolynomial > &factorisation, const carl::BitVector &reasons)
- GBPolynomial transformIntoEquality (ModuleInput::const_iterator constraint)
- GBPolynomial callGroebnerToSDP (const Ideal &gb)
- bool iterativeVariableRewriting ()
- bool findTrivialFactorisations ()
- void processNewConstraint (ModuleInput::const_iterator _formula)
- void handleConstraintToGBQueue (ModuleInput::const_iterator _formula)
- void handleConstraintNotToGB (ModuleInput::const_iterator _formula)
- void removeReceivedFormulaFromNewInequalities (ModuleInput::const_iterator _formula)
- void removeSubformulaFromPassedFormula (ModuleInput::iterator _formula)
- GBPolynomial rewriteVariable (const GBPolynomial &, const carl::Variable &, const TermT &, const BitVector &)
- bool validityCheck ()
- virtual bool informCore ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.
- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)
- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

Adds the given formula to the passed formula with no origin.

- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

  Copies the infeasible subsets of the passed formula.
- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.

- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- Settings::Groebner mBasis

  The current Groebner basis.
- InequalitiesTable< Settings > mInequalities

  The inequalities table for handling inequalities.
- std::vector< ModuleInput::const_iterator > mBacktrackPoints

  The vector of backtrack points, which has pointers to received constraints.
- std::list< GBModuleState< Settings > > mStateHistory

  Saves the relevant history to support backtracking.
- bool mRecalculateGB

  After popping in the history, it might be necessary to recalculate. This flag indicates this.
- std::list< typename InequalitiesTable< Settings >::Rows::iterator > mNewInequalities

  A list of inequalities which were added after the last consistency check.
- groebner::RewriteRules mRewriteRules

  The rewrite rules for the variables.
- std::map< ConstraintT, carl::Variable > mAdditionalVarMap
- FormulasT mGbEqualities

  A workaround to associate equalities in the passed formula originating from the gb (in contrast to those which originate from simplified formulae)
- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

Friends

- class InequalitiesTable< Settings >

0.15.211.1 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

0.15.211.2 Member Typedef Documentation

0.15.211.2.1 GBPolynomial `template<class Settings >`

```
typedef Settings::PolynomialWithReasons smtrat::GBModule< Settings >::GBPolynomial
```

0.15.211.2.2 Ideal `template<class Settings >`

```
typedef Settings::MultivariateIdeal smtrat::GBModule< Settings >::Ideal
```

0.15.211.2.3 Order `template<class Settings >`

```
typedef Settings::Order smtrat::GBModule< Settings >::Order
```

0.15.211.2.4 SettingsType `template<class Settings >`

```
typedef Settings smtrat::GBModule< Settings >::SettingsType
```

0.15.211.3 Member Enumeration Documentation

0.15.211.3.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

0.15.211.4 Constructor & Destructor Documentation

0.15.211.4.1 GBModule() `template<class Settings >`

```
smtrat::GBModule< Settings >::GBModule (
            const ModuleInput * const ,
            Conditionals & ,
            Manager * const  = NULL )
```

0.15.211.4.2 ~GBModule() `template<class Settings >`

```
virtual smtrat::GBModule< Settings >::~GBModule ( ) [virtual]
```

0.15.211.5 Member Function Documentation

0.15.211.5.1 add() `bool smtrat::Module::add (`

```
            ModuleInput::const_iterator _subformula ) [inherited]
```

The module has to take the given sub-formula of the received formula into account.

Parameters

_subformula	The sub-formula to take additionally into account.

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

0.15.211.5.2 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 >.

0.15.211.5.3 addCore() [1/2] `virtual bool smtrat::Module::addCore (`

```
[[maybe_unused] ] 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.

0.15.211.5.4 addCore() [2/2]

```
template<class Settings >
bool smtrat::GBModule< Settings >::addCore (
            ModuleInput::const_iterator _formula )
```

0.15.211.5.5 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula
(
            const FormulaT & formula )  [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.211.5.6 addLemma()

```
void smtrat::Module::addLemma (
            const FormulaT & _lemma,
            const LemmaType & _lt = LemmaType::NORMAL,
            const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) )  [inline],
[inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.211.5.7 addOrigin()

```
void smtrat::Module::addOrigin (
            ModuleInput::iterator _formula,
            const FormulaT & _origin )  [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.211.5.8 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
            ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.211.5.9 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.211.5.10 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula,
            const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.211.5.11 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.

0.15.211.5.12 allModels() `const std::vector<Model>& smtrat::Module::allModels ( ) const [inline], [inherited]`

Returns

All satisfying assignments, if existent.

0.15.211.5.13 anAnswerFound() `bool smtrat::Module::anAnswerFound ( ) const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.211.5.14 answerFound() `const smtrat::Conditionals& smtrat::Module::answerFound ( ) const [inline], [inherited]`

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.211.5.15 backendsModel() `const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.211.5.16 branchAt() [1/4] `bool smtrat::Module::branchAt (`

```
carl::Variable::Arg _var,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline],
[protected], [inherited]
```

0.15.211.5.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.211.5.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline],
[protected], [inherited]
```

0.15.211.5.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.211.5.20 callGroebnerToSDP()

```
template<class Settings >
GBPolynomial smtrat::GBModule< Settings >::callGroebnerToSDP (
    const Ideal & gb ) [protected]
```

0.15.211.5.21 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.

0.15.211.5.22 checkCore()

```
template<class Settings >
virtual Answer smtrat::GBModule< Settings >::checkCore ( )  [virtual]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.211.5.23 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.

0.15.211.5.24 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.

0.15.211.5.25 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.211.5.26 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

0.15.211.5.27 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

0.15.211.5.28 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

0.15.211.5.29 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.211.5.30 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

0.15.211.5.31 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.

0.15.211.5.32 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

0.15.211.5.33 constraintByGB()

```
template<class Settings >
bool smtrat::GBModule< Settings >::constraintByGB (
    carl::Relation cr ) [protected]
```

0.15.211.5.34 constraintsToInform() `const carl::FastSet<FormulaT>& smtrat::Module::constraints↩`

`ToInform ( ) const [inline], [inherited]`

Returns

The constraints which the backends must be informed about.

0.15.211.5.35 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

0.15.211.5.36 currentlySatisfiedByBackend()

`unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.211.5.37 deinform()

`void smtrat::Module::deinform ( const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.211.5.38 deinformCore()

`virtual void smtrat::Module::deinformCore ( [[maybe_unused] ] 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.

0.15.211.5.39 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

0.15.211.5.40 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 >.

0.15.211.5.41 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

0.15.211.5.42 factorisedConstraintDeduction()

```
template<class Settings >
void smtrat::GBModule< Settings >::factorisedConstraintDeduction (
            const std::list< GBPolynomial > & factorisation,
            const carl::BitVector & reasons ) [protected]
```

0.15.211.5.43 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
            const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

0.15.211.5.44 findTrivialFactorisations()

```
template<class Settings >
bool smtrat::GBModule< Settings >::findTrivialFactorisations ( ) [protected]
```

0.15.211.5.45 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.

0.15.211.5.46 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.

0.15.211.5.47 freeSplittingVariable() `static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.211.5.48 generateReasons() `template<class Settings > FormulasT smtrat::GBModule< Settings >::generateReasons ( const carl::BitVector & reasons ) [protected]`

0.15.211.5.49 generateReasonSet() `template<class Settings > FormulaSetT smtrat::GBModule< Settings >::generateReasonSet ( const carl::BitVector & reasons ) [protected]`

0.15.211.5.50 generateTrivialInfeasibleSubset() `void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.211.5.51 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.211.5.52 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.211.5.53 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.211.5.54 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.

0.15.211.5.55 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.211.5.56 getModelEqualities()

`std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.211.5.57 getOrigins() [1/3]

`void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.211.5.58 getOrigins() [2/3]

`void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.211.5.59 getOrigins() [3/3]

`const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]`

Gets the origins of the passed formula at the given position.

Parameters

_formula	The position of a formula in the passed formulas.

Returns

The origins of the passed formula at the given position.

0.15.211.5.60 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.

0.15.211.5.61 handleConstraintNotToGB() `template<class Settings > void smtrat::GBModule< Settings >::handleConstraintNotToGB ( ModuleInput::const_iterator _formula ) [protected]`

0.15.211.5.62 handleConstraintToGBQueue() `template<class Settings > void smtrat::GBModule< Settings >::handleConstraintToGBQueue ( ModuleInput::const_iterator _formula ) [protected]`

0.15.211.5.63 hasLemmas() `bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

0.15.211.5.64 hasValidInfeasibleSubset() `bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]`

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.211.5.65 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.211.5.66 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.211.5.67 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.

0.15.211.5.68 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.

0.15.211.5.69 informCore()

`virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.211.5.70 informedConstraints()

`const carl::FastSet<FormulaT>& smtrat::Module::informedConstraints ( ) const [inline], [inherited]`

Returns

The position of the first constraint of which no backend has been informed about.

0.15.211.5.71 init()

`void smtrat::Module::init ( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not be called twice and only before the first runBackends call.
Reimplemented in smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntBlastModule< Settings >, smtrat::FPPModule< Settings >, smtrat::CurryModule< Settings >, and smtrat::BVModule< Settings >.

0.15.211.5.72 is_minimizing()

`bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.211.5.73 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.

0.15.211.5.74 isPreprocessor()

`bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.211.5.75 iterativeVariableRewriting()

`template<class Settings > bool smtrat::GBModule< Settings >::iterativeVariableRewriting ( ) [protected]`

0.15.211.5.76 knownConstraintDeduction()

`template<class Settings > void smtrat::GBModule< Settings >::knownConstraintDeduction ( const std::list< std::pair< carl::BitVector, carl::BitVector > > & deductions ) [protected]`

0.15.211.5.77 lemmas()

`const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.211.5.78 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.

0.15.211.5.79 model()

`const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.211.5.80 modelsDisjoint()

```
bool smtrat::Module::modelsDisjoint (
    const Model & _modelA,
    const Model & _modelB ) [static], [protected], [inherited]
```

Checks whether there is no variable assigned by both models.

Parameters

_modelA	The first model to check for.
_modelB	The second model to check for.

Returns

true, if there is no variable assigned by both models; false, otherwise.

0.15.211.5.81 moduleName()

```
template<class Settings >
std::string smtrat::GBModule< Settings >::moduleName ( ) const [inline], [virtual]
```

Returns

The name of the given module type as name.

Reimplemented from smtrat::Module.

0.15.211.5.82 newConstraintDeduction()

```
template<class Settings >
void smtrat::GBModule< Settings >::newConstraintDeduction ( ) [protected]
```

0.15.211.5.83 objective()

```
carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]
```

0.15.211.5.84 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
    const FormulaT & _origin ) const [protected], [inherited]
```

0.15.211.5.85 passedFormulaBegin()

```
ModuleInput::iterator smtrat::Module::passedFormulaBegin (
) [inline], [protected], [inherited]
```

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.211.5.86 passedFormulaEnd()

```
ModuleInput::iterator smtrat::Module::passedFormulaEnd ( )
[inline], [protected], [inherited]
```

Returns

An iterator to the end of the passed formula.

0.15.211.5.87 passGB() template<class Settings >

void smtrat::GBModule< Settings >::passGB ( ) [protected]

0.15.211.5.88 popBacktrackPoint() template<class Settings >

void smtrat::GBModule< Settings >::popBacktrackPoint (ModuleInput::const_iterator *btpoint*) [protected]

0.15.211.5.89 pPassedFormula() const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]

Returns

A pointer to the formula passed to the backends of this module.

0.15.211.5.90 pReceivedFormula() const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]

Returns

A pointer to the formula passed to this module.

0.15.211.5.91 print() void smtrat::Module::print (const std::string & *_initiation* = "***") const [inherited]

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.211.5.92 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.

0.15.211.5.93 printInfeasibleSubsets() void smtrat::Module::printInfeasibleSubsets (const std::string & *_initiation* = "***") const [inherited]

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.211.5.94 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.

0.15.211.5.95 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.211.5.96 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.211.5.97 printRewriteRules()

```
template<class Settings >
void smtrat::GBModule< Settings >::printRewriteRules ( )
```

0.15.211.5.98 printStateHistory()

```
template<class Settings >
void smtrat::GBModule< Settings >::printStateHistory ( )
```

0.15.211.5.99 probablyLooping()

```
bool smtrat::Module::probablyLooping (
    const typename Poly::PolyType & _branchingPolynomial,
    const Rational & _branchingValue ) const [protected], [inherited]
```

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.211.5.100 processNewConstraint() template<class Settings >

```
void smtrat::GBModule< Settings >::processNewConstraint (
            ModuleInput::const_iterator _formula ) [protected]
```

0.15.211.5.101 pushBacktrackPoint() template<class Settings >

```
void smtrat::GBModule< Settings >::pushBacktrackPoint (
            ModuleInput::const_iterator btpoint ) [protected]
```

0.15.211.5.102 receivedFormulaChecked() void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]

Notifies that the received formulas has been checked.

0.15.211.5.103 receivedFormulasAsInfeasibleSubset() void smtrat::Module::receivedFormulasAs↩InfeasibleSubset (

```
            ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

0.15.211.5.104 receivedVariable() bool smtrat::Module::receivedVariable (

```
            carl::Variable::Arg _var ) const [inline], [inherited]
```

0.15.211.5.105 remove() void smtrat::Module::remove (

```
            ModuleInput::const_iterator _subformula ) [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.
However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

Parameters

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

0.15.211.5.106 removeCore() [1/2] virtual void smtrat::Module::removeCore (

```
            [[maybe_unused] ] 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.

0.15.211.5.107 removeCore() [2/2] template<class Settings >

```
void smtrat::GBModule< Settings >::removeCore (
            ModuleInput::const_iterator _formula )
```

0.15.211.5.108 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin (
            ModuleInput::iterator _formula,
            const FormulaT & _origin )  [inline], [protected], [inherited]
```

0.15.211.5.109 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins )  [inline], [protected], [inherited]
```

0.15.211.5.110 removeReceivedFormulaFromNewInequalities()

```
template<class Settings >
void smtrat::GBModule< Settings >::removeReceivedFormulaFromNewInequalities (
            ModuleInput::const_iterator _formula )  [protected]
```

0.15.211.5.111 removeSubformulaFromPassedFormula()

```
template<class Settings >
void smtrat::GBModule< Settings >::removeSubformulaFromPassedFormula (
            ModuleInput::iterator _formula )  [protected]
```

0.15.211.5.112 rewriteVariable()

```
template<class Settings >
GBPolynomial smtrat::GBModule< Settings >::rewriteVariable (
            const GBPolynomial & ,
            const carl::Variable & ,
            const TermT & ,
            const BitVector &  )  [inline], [protected]
```

0.15.211.5.113 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const  [inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.211.5.114 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const  [inline], [inherited]
```

Returns

A reference to the formula passed to this module.

0.15.211.5.115 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( )  [inline], [protected], [virtual], [inherited]
```

Reimplemented in smtrat::PModule.

0.15.211.5.116 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.

0.15.211.5.117 saveState()

`template<class Settings >`
`bool smtrat::GBModule< Settings >::saveState ( ) [protected]`

0.15.211.5.118 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.

0.15.211.5.119 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.

0.15.211.5.120 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.

0.15.211.5.121 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 !=.

0.15.211.5.122 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.211.5.123 transformIntoEquality() `template<class Settings > GBPolynomial smtrat::GBModule< Settings >::transformIntoEquality ( ModuleInput::const_iterator constraint ) [protected]`

0.15.211.5.124 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 >.

0.15.211.5.125 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.211.5.126 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::PModule, smtrat::VSModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::STropModule< Settings >, smtrat::SATModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::MCBModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntEqModule< Settings >, smtrat::IntBlastModule< Settings >, smtrat::IncWidthModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::GBPPModule< Settings >, smtrat::FPPModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::ESModule< Settings >, smtrat::CurryModule< Settings >, smtrat::CubeLIAModule< Se
smtrat::CSplitModule< Settings >, smtrat::CoveringNGModule< Settings >, and smtrat::BVModule< Settings >.

0.15.211.5.127 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).

0.15.211.5.128 validityCheck() `template<class Settings > bool smtrat::GBModule< Settings >::validityCheck ( ) [protected]`

0.15.211.6 Friends And Related Function Documentation

0.15.211.6.1 InequalitiesTable< Settings > template<class Settings >

friend class InequalitiesTable< Settings > [friend]

0.15.211.7 Field Documentation

0.15.211.7.1 mAdditionalVarMap template<class Settings >

std::map<ConstraintT, carl::Variable> smtrat::GBModule< Settings >::mAdditionalVarMap [protected]

0.15.211.7.2 mAllBackends std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]

The backends of this module which have been used.

0.15.211.7.3 mAllModels std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]

Stores all satisfying assignments.

0.15.211.7.4 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.

0.15.211.7.5 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.

0.15.211.7.6 mBasis template<class Settings >

Settings::Groebner smtrat::GBModule< Settings >::mBasis [protected]

The current Groebner basis.

0.15.211.7.7 mConstraintsToInform carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]

Stores the constraints which the backends must be informed about.

0.15.211.7.8 mFinalCheck bool smtrat::Module::mFinalCheck [protected], [inherited]

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.211.7.9 mFirstPosInLastBranches std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]

The beginning of the cyclic buffer storing the last branches.

0.15.211.7.10 mFirstSubformulaToPass ModuleInput::iterator smtrat::Module::mFirstSubformula↩ ToPass [protected], [inherited]

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.211.7.11 mFirstUncheckedReceivedSubformula

`ModuleInput::const_iterator smtrat::Module↩::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.211.7.12 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.211.7.13 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.211.7.14 mGbEqualities

`template<class Settings > FormulasT smtrat::GBModule< Settings >::mGbEqualities [protected]`

A workaround to associate equalities in the passed formula originating from the gb (in contrast to those which originate from simplified formulae)

0.15.211.7.15 mInequalities

`template<class Settings > InequalitiesTable<Settings> smtrat::GBModule< Settings >::mInequalities [protected]`

The inequalities table for handling inequalities.

0.15.211.7.16 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.211.7.17 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.211.7.18 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std↩::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.211.7.19 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.211.7.20 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.211.7.21 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.211.7.22 mNewInequalities

`template<class Settings > std::list<typename InequalitiesTable<Settings>::Rows::iterator> smtrat::GBModule< Settings >::mNewInequalities [protected]`

A list of inequalities which were added after the last consistency check.

0.15.211.7.23 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.211.7.24 mObjectiveVariable

`carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.211.7.25 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

0.15.211.7.26 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.211.7.27 mRecalculateGB

`template<class Settings > bool smtrat::GBModule< Settings >::mRecalculateGB [protected]`

After popping in the history, it might be necessary to recalculate. This flag indicates this.

0.15.211.7.28 mRewriteRules

`template<class Settings > groebner::RewriteRules smtrat::GBModule< Settings >::mRewriteRules [protected]`

The rewrite rules for the variables.

0.15.211.7.29 mSmallerMusesCheckCounter

`unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.211.7.30 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.

0.15.211.7.31 mStateHistory

`template<class Settings > std::list<GBModuleState<Settings> > smtrat::GBModule< Settings >::mStateHistory [protected]`

Saves the relevant history to support backtracking.

0.15.211.7.32 mTheoryPropagations

`std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

0.15.211.7.33 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).

0.15.211.7.34 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.212 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

0.15.212.1 Detailed Description

template<typename Settings>
class smtrat::GBModuleState< Settings >

A class to save the current state of a GBModule.
Used for backtracking-support

0.15.212.2 Constructor & Destructor Documentation

0.15.212.2.1 GBModuleState() [1/2]

```
template<typename Settings >
smtrat::GBModuleState< Settings >::GBModuleState ( ) [inline]
```

0.15.212.2.2 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]
```

0.15.212.3 Member Function Documentation

0.15.212.3.1 getBasis()

```
template<typename Settings >
const Settings::Groebner& smtrat::GBModuleState< Settings >::getBasis ( ) const [inline]
```

0.15.212.3.2 getRewriteRules() `template<typename Settings >`

`const std::map<carl::Variable, std::pair<TermT, carl::BitVector> >& smtrat::GBModuleState< Settings >::getRewriteRules ( ) const [inline]`

0.15.212.4 Field Documentation

0.15.212.4.1 mBasis `template<typename Settings >`

`const Settings::Groebner smtrat::GBModuleState< Settings >::mBasis [protected]`

The state of the basis.

0.15.212.4.2 mRewrites `template<typename Settings >`

`const std::map<carl::Variable, std::pair<TermT, carl::BitVector> > smtrat::GBModuleState< Settings >::mRewrites [protected]`

0.15.213 smtrat::GBPPModule< Settings > Class Template Reference

`#include <GBPPModule.h>`

Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- GBPPModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=nullptr)
- ~GBPPModule ()
- void updateModel () const

  Updates the current assignment into the model.
- Answer checkCore ()

  Checks the received formula for consistency.
- bool isPreprocessor () const
- bool appliedPreprocessing () const
- bool add (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- void remove (ModuleInput::const_iterator _subformula)

  Removes everything related to the given sub-formula of the received formula.
- Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)

  Checks the received formula for consistency.
- Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- std::pair< bool, FormulaT > getReceivedFormulaSimplified ()
- 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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] 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 >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.

- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.213.1 Member Typedef Documentation

0.15.213.1.1 SettingsType

```
template<typename Settings >
typedef Settings smtrat::GBPPModule< Settings >::SettingsType
```

0.15.213.2 Member Enumeration Documentation

0.15.213.2.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

0.15.213.3 Constructor & Destructor Documentation

0.15.213.3.1 GBPPModule()

```
template<typename Settings >
smtrat::GBPPModule< Settings >::GBPPModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

0.15.213.3.2 ~GBPPModule()

```
template<typename Settings >
smtrat::GBPPModule< Settings >::~GBPPModule ( )
```

0.15.213.4 Member Function Documentation

0.15.213.4.1 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.

0.15.213.4.2 addConstraintToInform()

```
void smtrat::Module::addConstraintToInform (
    const FormulaT & _constraint ) [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

0.15.213.4.3 addCore()

```
virtual bool smtrat::Module::addCore (
    [[maybe_unused] ] 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.

0.15.213.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula
(
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.213.4.5 addLemma()

```
void smtrat::Module::addLemma (
    const FormulaT & _lemma,
    const LemmaType & _lt = LemmaType::NORMAL,
    const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline],
[inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.213.4.6 addOrigin()

```
void smtrat::Module::addOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.213.4.7 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.213.4.8 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.213.4.9 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.213.4.10 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.

0.15.213.4.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns

All satisfying assignments, if existent.

0.15.213.4.12 anAnswerFound()

```
bool smtrat::Module::anAnswerFound ( ) const [inline], [protected],
[inherited]
```

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.213.4.13 answerFound()

```
const smtrat::Conditionals& smtrat::Module::answerFound ( ) const
[inline], [inherited]
```

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.213.4.14 appliedPreprocessing()

`bool smtrat::PModule::appliedPreprocessing ( ) const [inline], [inherited]`

Returns

true, if the current received formula has been simplified and the result of this simplification is stored in the passed formula.

0.15.213.4.15 backendsModel()

`const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.213.4.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.213.4.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.213.4.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.213.4.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.213.4.20 check()

```
Answer smtrat::PModule::check (
    bool _final = false,
    bool _full = true,
    carl::Variable _objective = carl::Variable::NO_VARIABLE ) [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.213.4.21 checkCore()

```
template<typename Settings >
Answer smtrat::GBPPModule< Settings >::checkCore ( ) [virtual]
```

Checks the received formula for consistency.

Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.213.4.22 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.

0.15.213.4.23 checkModel() `unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]`

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.213.4.24 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.213.4.25 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

0.15.213.4.26 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

0.15.213.4.27 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

0.15.213.4.28 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.213.4.29 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

0.15.213.4.30 collectOrigins() [2/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.
If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.213.4.31 collectTheoryPropagations()

`void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

0.15.213.4.32 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ToInform ( ) const [inline], [inherited]`

Returns

The constraints which the backends must be informed about.

0.15.213.4.33 currentlySatisfied()

`virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.213.4.34 currentlySatisfiedByBackend()

`unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.213.4.35 deinform()

`void smtrat::Module::deinform ( const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.213.4.36 deinformCore()

`virtual void smtrat::Module::deinformCore ( [[maybe_unused] ] 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.

0.15.213.4.37 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

0.15.213.4.38 eraseSubformulaFromPassedFormula()

```
ModuleInput::iterator smtrat::Module::erase↩
SubformulaFromPassedFormula (
    ModuleInput::iterator _subformula,
    bool _ignoreOrigins = false ) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_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 >.

0.15.213.4.39 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceived↩
VariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

0.15.213.4.40 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::find↩
BestOrigin (
    const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

0.15.213.4.41 firstSubformulaToPass() `ModuleInput::const_iterator smtrat::Module::firstSubformula↩ ToPass ( ) const [inline], [inherited]`

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.213.4.42 firstUncheckedReceivedSubformula() `ModuleInput::const_iterator smtrat::Module↩ ::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.213.4.43 freeSplittingVariable() `static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.213.4.44 generateTrivialInfeasibleSubset() `void smtrat::Module::generateTrivialInfeasible↩ Subset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.213.4.45 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.213.4.46 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.213.4.47 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.213.4.48 getInfeasibleSubsets() [2/2] `std::vector< FormulaSetT > smtrat::Module::get↩ InfeasibleSubsets ( 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.

0.15.213.4.49 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.213.4.50 getModelEqualities()

`std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.213.4.51 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.213.4.52 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.213.4.53 getOrigins() [3/3]

```
const FormulaT& smtrat::Module::getOrigins (
    ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]
```

Gets the origins of the passed formula at the given position.

Parameters

_formula	The position of a formula in the passed formulas.

Returns

The origins of the passed formula at the given position.

0.15.213.4.54 getReceivedFormulaSimplified() `std::pair< bool, FormulaT > smtrat::PModule::getReceivedFormulaSimplified ( ) [virtual], [inherited]`

Returns

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's reveived formula, if the Boolean is true.

Reimplemented from smtrat::Module.

0.15.213.4.55 hasLemmas() `bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

0.15.213.4.56 hasValidInfeasibleSubset() `bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]`

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.213.4.57 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.213.4.58 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.213.4.59 inform() `bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.
It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

Parameters

_constraint	The constraint to inform about.

Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

0.15.213.4.60 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.

0.15.213.4.61 informCore()

`virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.213.4.62 informedConstraints()

`const carl::FastSet<FormulaT>& smtrat::Module::informedConstraints ( ) const [inline], [inherited]`

Returns

The position of the first constraint of which no backend has been informed about.

0.15.213.4.63 init()

`void smtrat::Module::init ( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not be called twice and only before the first runBackends call.
Reimplemented in smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntBlastModule< Settings >, smtrat::FPPModule< Settings >, smtrat::CurryModule< Settings >, and smtrat::BVModule< Settings >.

0.15.213.4.64 is_minimizing()

`bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.213.4.65 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.

0.15.213.4.66 isPreprocessor() `bool smtrat::PModule::isPreprocessor ( ) const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.213.4.67 lemmas() `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.213.4.68 merge() `std::vector< FormulaT > smtrat::Module::merge ( const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.
({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

Parameters

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns

The vector being the two given vectors merged.

0.15.213.4.69 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.213.4.70 modelsDisjoint() `bool smtrat::Module::modelsDisjoint ( const Model & _modelA, const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

Parameters

_modelA	The first model to check for.
_modelB	The second model to check for.

Returns

true, if there is no variable assigned by both models; false, otherwise.

0.15.213.4.71 moduleName()

`virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]`

Returns

The name of the given module type as name.

Reimplemented in smtrat::VSModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::STropModule< Settings >, smtrat::SplitSOSModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntEqModule< Settings >, smtrat::IntBlastModule< Settings >, smtrat::IncWidthModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::GBModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::CurryModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::BVModule< Settings >, and smtrat::BEModule< Settings >.

0.15.213.4.72 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

0.15.213.4.73 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.213.4.74 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.213.4.75 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.213.4.76 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.213.4.77 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.213.4.78 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.213.4.79 printAllModels()

`void smtrat::Module::printAllModels ( std::ostream & _out = std::cout ) [inherited]`

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.213.4.80 printInfeasibleSubsets()

`void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.213.4.81 printModel()

`void smtrat::Module::printModel ( std::ostream & _out = std::cout ) const [inherited]`

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.213.4.82 printPassedFormula()

`void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.213.4.83 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.213.4.84 probablyLooping()

`bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.213.4.85 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.213.4.86 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

0.15.213.4.87 receivedVariable()

`bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

0.15.213.4.88 remove()

`void smtrat::PModule::remove ( ModuleInput::const_iterator _subformula ) [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.
However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

Parameters

_subformula	The sub formula of the received formula to remove.

Reimplemented from smtrat::Module.

0.15.213.4.89 removeCore()

`virtual void smtrat::Module::removeCore ( [[maybe_unused] ] 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.

0.15.213.4.90 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin (
            ModuleInput::iterator _formula,
            const FormulaT & _origin ) [inline], [protected], [inherited]
```

0.15.213.4.91 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]
```

0.15.213.4.92 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.213.4.93 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]
```

Returns

A reference to the formula passed to this module.

0.15.213.4.94 runBackends() [1/2]

```
virtual Answer smtrat::PModule::runBackends ( ) [inline], [virtual], [inherited]
```

Reimplemented from smtrat::Module.

0.15.213.4.95 runBackends() [2/2]

```
Answer smtrat::PModule::runBackends (
            bool _final,
            bool _full,
            carl::Variable _objective ) [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

Parameters

_final	true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT.
_full	false, if this module should avoid too expensive procedures and rather return unknown instead.
_objective	if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good.

Returns

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.213.4.96 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.

0.15.213.4.97 setThreadPriority()

`void smtrat::Module::setThreadPriority ( thread_priority _threadPriority ) [inline], [inherited]`

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.213.4.98 solverState()

`Answer smtrat::Module::solverState ( ) const [inline], [inherited]`

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.213.4.99 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 !=.

0.15.213.4.100 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.213.4.101 updateAllModels() `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.213.4.102 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.213.4.103 updateModel() `template<typename Settings > void smtrat::GBPPModule< Settings >::updateModel ( ) const [virtual]`

Updates the current assignment into the model.
Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.
Reimplemented from smtrat::Module.

0.15.213.4.104 usedBackends() `const std::vector<Module*>& smtrat::Module::usedBackends ( ) const [inline], [inherited]`

Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.213.5 Field Documentation

0.15.213.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.213.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.213.5.3 mBackendsFoundAnswer `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.213.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.213.5.5 mFinalCheck `bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.213.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.213.5.7 mFirstSubformulaToPass

`ModuleInput::iterator smtrat::Module::mFirstSubformulaToPass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.213.5.8 mFirstUncheckedReceivedSubformula

`ModuleInput::const_iterator smtrat::Module::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.213.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.213.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.213.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.213.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.213.5.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.213.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.213.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.213.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.213.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.213.5.18 mObjectiveVariable `carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.213.5.19 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]`

Reusable splitting variables.

0.15.213.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.213.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.213.5.22 mSolverState `std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.213.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]`

0.15.213.5.24 mUsedBackends `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.213.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.214 Minisat::Hash< K > Struct Template Reference

`#include <Map.h>`

Public Member Functions

- uint32_t operator() (const K &k) const

0.15.214.1 Member Function Documentation

0.15.214.1.1 operator()() `template<class K > uint32_t Minisat::Hash< K >::operator() ( const K & k ) const [inline]`

0.15.215 std::hash< const smtrat::expression::ExpressionContent * > Struct Reference

`#include <ExpressionHash.h>`

Public Member Functions

- std::size_t operator() (const smtrat::expression::ExpressionContent *ec) const

0.15.215.1 Member Function Documentation

0.15.215.1.1 operator()() `std::size_t std::hash< const smtrat::expression::ExpressionContent * >::operator() (`
`const smtrat::expression::ExpressionContent * ec ) const [inline]`

0.15.216 std::hash< smtrat::expression::BinaryExpression > Struct Reference

`#include <ExpressionHash.h>`

Public Member Functions

- std::size_t operator() (const smtrat::expression::BinaryExpression &ec) const

0.15.216.1 Member Function Documentation

0.15.216.1.1 operator()() `std::size_t std::hash< smtrat::expression::BinaryExpression >::operator() (`
`const smtrat::expression::BinaryExpression & ec ) const [inline]`

0.15.217 std::hash< smtrat::expression::Expression > Struct Reference

`#include <ExpressionHash.h>`

Public Member Functions

- std::size_t operator() (const smtrat::expression::Expression &expr) const

0.15.217.1 Member Function Documentation

0.15.217.1.1 operator()() `std::size_t std::hash< smtrat::expression::Expression >::operator() (`
`const smtrat::expression::Expression & expr ) const [inline]`

0.15.218 std::hash< smtrat::expression::ITEExpression > Struct Reference

`#include <ExpressionHash.h>`

Public Member Functions

- std::size_t operator() (const smtrat::expression::ITEExpression &ec) const

0.15.218.1 Member Function Documentation

0.15.218.1.1 operator()()

```
std::size_t std::hash< smtrat::expression::ITEExpression >::operator()
(
            const smtrat::expression::ITEExpression & ec ) const  [inline]
```

0.15.219 std::hash< smtrat::expression::NaryExpression > Struct Reference

```
#include <ExpressionHash.h>
```

Public Member Functions

- std::size_t operator() (const smtrat::expression::NaryExpression &ec) const

0.15.219.1 Member Function Documentation

0.15.219.1.1 operator()()

```
std::size_t std::hash< smtrat::expression::NaryExpression >::operator()
(
            const smtrat::expression::NaryExpression & ec ) const  [inline]
```

0.15.220 std::hash< smtrat::expression::QuantifierExpression > Struct Reference

```
#include <ExpressionHash.h>
```

Public Member Functions

- std::size_t operator() (const smtrat::expression::QuantifierExpression &ec) const

0.15.220.1 Member Function Documentation

0.15.220.1.1 operator()()

```
std::size_t std::hash< smtrat::expression::QuantifierExpression >↩
::operator() (
            const smtrat::expression::QuantifierExpression & ec ) const  [inline]
```

0.15.221 std::hash< smtrat::expression::UnaryExpression > Struct Reference

```
#include <ExpressionHash.h>
```

Public Member Functions

- std::size_t operator() (const smtrat::expression::UnaryExpression &ec) const

0.15.221.1 Member Function Documentation

0.15.221.1.1 operator()()

```
std::size_t std::hash< smtrat::expression::UnaryExpression >::operator()
(
            const smtrat::expression::UnaryExpression & ec ) const  [inline]
```

0.15.222 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

0.15.222.1 Detailed Description

template<typename T1, typename T2>
struct std::hash< smtrat::lra::Variable< T1, T2 > >

Implements std::hash for sort value.

0.15.222.2 Member Function Documentation

0.15.222.2.1 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.

0.15.223 std::hash< smtrat::vs::Substitution > Struct Reference

```
#include <Substitution.h>
```

Public Member Functions

- size_t operator() (const smtrat::vs::Substitution &_substitution) const

0.15.223.1 Member Function Documentation

0.15.223.1.1 operator()()

```
size_t std::hash< smtrat::vs::Substitution >::operator() (
            const smtrat::vs::Substitution & _substitution ) const  [inline]
```

0.15.224 std::hash< std::vector< T > > Struct Template Reference

```
#include <ExpressionHash.h>
```

Public Member Functions

- std::size_t operator() (const std::vector< T > &v) const

0.15.224.1 Member Function Documentation

0.15.224.1.1 operator()()

```
template<typename T >
std::size_t std::hash< std::vector< T > >::operator() (
            const std::vector< T > & v ) const  [inline]
```

0.15.225 std::hash_combiner Struct Reference

`#include <ExpressionHash.h>`

Public Member Functions

- hash_combiner (std::size_t _seed=0)
- hash_combiner & operator() (std::size_t h)
- template<typename T >
  hash_combiner & operator() (const T &t)
- operator std::size_t () const

Data Fields

- std::size_t seed

0.15.225.1 Constructor & Destructor Documentation

0.15.225.1.1 hash_combiner() `std::hash_combiner::hash_combiner ( std::size_t _seed = 0 ) [inline]`

0.15.225.2 Member Function Documentation

0.15.225.2.1 operator std::size_t() `std::hash_combiner::operator std::size_t ( ) const [inline]`

0.15.225.2.2 operator()() [1/2] `template<typename T > hash_combiner& std::hash_combiner::operator() ( const T & t ) [inline]`

0.15.225.2.3 operator()() [2/2] `hash_combiner& std::hash_combiner::operator() ( std::size_t h ) [inline]`

0.15.225.3 Field Documentation

0.15.225.3.1 seed `std::size_t std::hash_combiner::seed`

0.15.226 Minisat::Heap< Comp > Class Template Reference

`#include <Heap.h>`

Public Member Functions

- Heap (const Comp &c)
- int size () const
- bool empty () const
- bool inHeap (int n) const
- int operator[] (int index) const
- void decrease (int n)
- void increase (int n)
- void update (int n)

- void insert (int n)
- int removeMin ()
- void build (vec< int > &ns)
- void clear (bool dealloc=false)
- void print () const

0.15.226.1 Constructor & Destructor Documentation

0.15.226.1.1 Heap()

```
template<class Comp >
Minisat::Heap< Comp >::Heap (
            const Comp & c ) [inline]
```

0.15.226.2 Member Function Documentation

0.15.226.2.1 build()

```
template<class Comp >
void Minisat::Heap< Comp >::build (
            vec< int > & ns ) [inline]
```

0.15.226.2.2 clear()

```
template<class Comp >
void Minisat::Heap< Comp >::clear (
            bool dealloc = false ) [inline]
```

0.15.226.2.3 decrease()

```
template<class Comp >
void Minisat::Heap< Comp >::decrease (
            int n ) [inline]
```

0.15.226.2.4 empty()

```
template<class Comp >
bool Minisat::Heap< Comp >::empty ( ) const [inline]
```

0.15.226.2.5 increase()

```
template<class Comp >
void Minisat::Heap< Comp >::increase (
            int n ) [inline]
```

0.15.226.2.6 inHeap()

```
template<class Comp >
bool Minisat::Heap< Comp >::inHeap (
            int n ) const [inline]
```

0.15.226.2.7 insert()

```
template<class Comp >
void Minisat::Heap< Comp >::insert (
            int n ) [inline]
```

0.15.226.2.8 operator[]()

```
template<class Comp >
int Minisat::Heap< Comp >::operator[] (
            int index ) const [inline]
```

0.15.226.2.9 print() `template<class Comp >`

`void Minisat::Heap< Comp >::print ( ) const [inline]`

0.15.226.2.10 removeMin() `template<class Comp >`

`int Minisat::Heap< Comp >::removeMin ( ) [inline]`

0.15.226.2.11 size() `template<class Comp >`

`int Minisat::Heap< Comp >::size ( ) const [inline]`

0.15.226.2.12 update() `template<class Comp >`

```
void Minisat::Heap< Comp >::update (
            int n ) [inline]
```

0.15.227 smtrat::parser::HexadecimalParser Struct Reference

Parses hexadecimals: `#x[0-9a-fA-F]+`

`#include <Lexicon.h>`

Public Types

- typedef boost::iterator_range< Iterator > ITRange

Public Member Functions

- HexadecimalParser ()
- FixedWidthConstant< Integer > build (const ITRange &itr, const Integer &val)

Data Fields

- qi::uint_parser< Integer, 16, 1,-1 > number
- qi::rule< Iterator, FixedWidthConstant< Integer >), Skipper, qi::locals< Integer > > main
- qi::rule< Iterator, FixedWidthConstant< Integer >), Skipper > main2

0.15.227.1 Detailed Description

Parses hexadecimals: `#x[0-9a-fA-F]+`

0.15.227.2 Member Typedef Documentation

0.15.227.2.1 ITRange `typedef boost::iterator_range<Iterator> smtrat::parser::HexadecimalParser::ITRange`

0.15.227.3 Constructor & Destructor Documentation

0.15.227.3.1 HexadecimalParser() `smtrat::parser::HexadecimalParser::HexadecimalParser ( ) [inline]`

0.15.227.4 Member Function Documentation

0.15.227.4.1 build()

```
FixedWidthConstant<Integer> smtrat::parser::HexadecimalParser::build (
    const ITRange & itr,
    const Integer & val ) [inline]
```

0.15.227.5 Field Documentation

0.15.227.5.1 main

```
qi::rule<Iterator, FixedWidthConstant<Integer>), Skipper, qi::locals<Integer>
> smtrat::parser::HexadecimalParser::main
```

0.15.227.5.2 main2

```
qi::rule<Iterator, FixedWidthConstant<Integer>), Skipper> smtrat::parser↩
::HexadecimalParser::main2
```

0.15.227.5.3 number

```
qi::uint_parser<Integer,16,1,-1> smtrat::parser::HexadecimalParser↩
::number
```

0.15.228 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 ()

0.15.228.1 Constructor & Destructor Documentation

0.15.228.1.1 HistoryNode() [1/2] `smtrat::icp::HistoryNode::HistoryNode ( ) [inline]`

Methods.

0.15.228.1.2 HistoryNode() [2/2] `smtrat::icp::HistoryNode::HistoryNode ( const EvalDoubleIntervalMap & _intervals ) [inline]`

0.15.228.1.3 ~HistoryNode() `smtrat::icp::HistoryNode::~HistoryNode ( ) [inline]`

0.15.228.2 Member Function Documentation

0.15.228.2.1 addContraction() `void smtrat::icp::HistoryNode::addContraction ( ContractionCandidate * _candidate, const set_icpVariable & _variables ) [inline]`

0.15.228.2.2 addInfeasibleConstraint() `bool smtrat::icp::HistoryNode::addInfeasibleConstraint ( const ConstraintT & _constraint, bool _addOnlyConstraint = false ) [inline]`

0.15.228.2.3 addInfeasibleVariable() `bool smtrat::icp::HistoryNode::addInfeasibleVariable ( const IcpVariable * _variable, bool _addOnlyVariable = false ) [inline]`

0.15.228.2.4 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

0.15.228.2.5 addReason()

```
void smtrat::icp::HistoryNode::addReason (
    carl::Variable::Arg _variable,
    const ConstraintT & _reason ) [inline]
```

0.15.228.2.6 addReasons() [1/2]

```
void smtrat::icp::HistoryNode::addReasons (
    carl::Variable::Arg _variable,
    const FormulasT & _origins ) [inline]
```

0.15.228.2.7 addReasons() [2/2]

```
void smtrat::icp::HistoryNode::addReasons (
    carl::Variable::Arg _variable,
    const std::set< ConstraintT > & _reasons ) [inline]
```

0.15.228.2.8 addVariableReason()

```
void smtrat::icp::HistoryNode::addVariableReason (
    carl::Variable::Arg _variable,
    const IcpVariable * _reason ) [inline]
```

0.15.228.2.9 addVariableReasons()

```
void smtrat::icp::HistoryNode::addVariableReasons (
    carl::Variable::Arg _variable,
    set_icpVariable _variables ) [inline]
```

0.15.228.2.10 appliedConstraints() [1/2]

```
FormulaSetT smtrat::icp::HistoryNode::appliedConstraints
( ) [inline]
```

0.15.228.2.11 appliedConstraints() [2/2]

```
void smtrat::icp::HistoryNode::appliedConstraints (
    std::vector< FormulaT > & _result ) [inline]
```

0.15.228.2.12 appliedContractions()

```
std::set<const ContractionCandidate*> smtrat::icp::History↩
Node::appliedContractions ( ) [inline]
```

0.15.228.2.13 getCandidates()

```
const std::set<const ContractionCandidate*>& smtrat::icp::↩
HistoryNode::getCandidates ( ) const [inline]
```

0.15.228.2.14 getInterval()

```
DoubleInterval& smtrat::icp::HistoryNode::getInterval (
    carl::Variable::Arg _variable ) [inline]
```

0.15.228.2.15 hasEmptyInterval()

```
bool smtrat::icp::HistoryNode::hasEmptyInterval ( ) const
[inline]
```

0.15.228.2.16 intervals()

`const EvalDoubleIntervalMap& smtrat::icp::HistoryNode::intervals ( ) const [inline]`

Getters and Setters.

0.15.228.2.17 print()

`void smtrat::icp::HistoryNode::print ( std::ostream & _out = std::cout ) const [inline]`

0.15.228.2.18 printReasons()

`void smtrat::icp::HistoryNode::printReasons ( std::ostream & _out = std::cout ) const [inline]`

0.15.228.2.19 printVariableReasons()

`void smtrat::icp::HistoryNode::printVariableReasons ( std::ostream & _out = std::cout ) const [inline]`

0.15.228.2.20 propagateStateInfeasibleConstraints()

`void smtrat::icp::HistoryNode::propagateStateInfeasibleConstraints ( HistoryNode * _target ) const [inline]`

0.15.228.2.21 propagateStateInfeasibleVariables()

`void smtrat::icp::HistoryNode::propagateStateInfeasibleVariables ( HistoryNode * _target ) const [inline]`

0.15.228.2.22 reasons() [1/2]

`const std::map<carl::Variable, std::set<ConstraintT> >& smtrat::icp::HistoryNode::reasons ( ) const [inline]`

0.15.228.2.23 reasons() [2/2]

`std::set<ConstraintT>& smtrat::icp::HistoryNode::reasons ( carl::Variable::Arg _variable ) [inline]`

0.15.228.2.24 reset()

`void smtrat::icp::HistoryNode::reset ( ) [inline]`

0.15.228.2.25 rIntervals()

`EvalDoubleIntervalMap& smtrat::icp::HistoryNode::rIntervals ( ) [inline]`

0.15.228.2.26 rReasons()

`std::map<carl::Variable, std::set<ConstraintT> >& smtrat::icp::HistoryNode::rReasons ( ) [inline]`

0.15.228.2.27 rStateInfeasibleConstraints()

`std::set<ConstraintT>& smtrat::icp::HistoryNode::rStateInfeasibleConstraints ( ) [inline]`

0.15.228.2.28 rStateInfeasibleVariables()

`set_icpVariable& smtrat::icp::HistoryNode::rStateInfeasibleVariables ( ) [inline]`

0.15.228.2.29 rVariableReasons()

`std::map<carl::Variable, set_icpVariable>& smtrat::icp::HistoryNode::rVariableReasons ( ) [inline]`

0.15.228.2.30 setIntervals()

`void smtrat::icp::HistoryNode::setIntervals ( const EvalDoubleIntervalMap & _map ) [inline]`

0.15.228.2.31 stateInfeasibleConstraints()

`const std::set<ConstraintT>& smtrat::icp::HistoryNode::stateInfeasibleConstraints ( ) const [inline]`

0.15.228.2.32 stateInfeasibleVariables()

`set_icpVariable smtrat::icp::HistoryNode::stateInfeasibleVariables ( ) const [inline]`

0.15.228.2.33 variableHull()

`void smtrat::icp::HistoryNode::variableHull ( carl::Variable::Arg _variable, set_icpVariable & _result ) const [inline]`

0.15.228.2.34 variableReasons() [1/2]

`const std::map<carl::Variable, set_icpVariable>& smtrat::icp::HistoryNode::variableReasons ( ) [inline]`

0.15.228.2.35 variableReasons() [2/2]

`set_icpVariable smtrat::icp::HistoryNode::variableReasons ( carl::Variable::Arg _variable ) [inline]`

0.15.229 smtrat::ICEModule< Settings > Class Template Reference

`#include <ICEModule.h>`

Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- ICEModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=nullptr)
- ~ICEModule ()
- bool addCore (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- void removeCore (ModuleInput::const_iterator _subformula)

  Removes the subformula of the received formula at the given position to the considered ones of this module.
- Answer checkCore ()

  Checks the received formula for consistency.
- bool isPreprocessor () const
- bool appliedPreprocessing () const
- bool add (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- void remove (ModuleInput::const_iterator _subformula)

  Removes everything related to the given sub-formula of the received formula.
- Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)

Checks the received formula for consistency.

- Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- std::pair< bool, FormulaT > getReceivedFormulaSimplified ()
- 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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

Removes everything related to the given sub-formula of the received formula.

- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.
- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)
- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

  Copies the infeasible subsets of the passed formula.
- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.

- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

false, if this module should avoid too expensive procedures and rather return unknown instead.

- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.229.1 Member Typedef Documentation

0.15.229.1.1 SettingsType

```
template<typename Settings >
typedef Settings smtrat::ICEModule< Settings >::SettingsType
```

0.15.229.2 Member Enumeration Documentation

0.15.229.2.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

0.15.229.3 Constructor & Destructor Documentation

0.15.229.3.1 ICEModule() template<typename Settings >

```
smtrat::ICEModule< Settings >::ICEModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

0.15.229.3.2 ~ICEModule() template<typename Settings >

```
smtrat::ICEModule< Settings >::~ICEModule ( )
```

0.15.229.4 Member Function Documentation

0.15.229.4.1 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.

0.15.229.4.2 addConstraintToInform()

```
void smtrat::Module::addConstraintToInform (
            const FormulaT & _constraint ) [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

0.15.229.4.3 addCore() [1/2]

```
virtual bool smtrat::Module::addCore (
            [[maybe_unused] ] 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.

0.15.229.4.4 addCore() [2/2] template<typename Settings >

```
bool smtrat::ICEModule< Settings >::addCore (
            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.

0.15.229.4.5 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
            const FormulaT & formula )  [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.229.4.6 addLemma()

```
void smtrat::Module::addLemma (
            const FormulaT & _lemma,
            const LemmaType & _lt = LemmaType::NORMAL,
            const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) )  [inline],
[inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.229.4.7 addOrigin()

```
void smtrat::Module::addOrigin (
            ModuleInput::iterator _formula,
            const FormulaT & _origin )  [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.229.4.8 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
            ModuleInput::const_iterator _subformula )  [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.229.4.9 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula )  [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.229.4.10 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula,
            const FormulaT & _origin )  [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.229.4.11 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.

0.15.229.4.12 allModels() `const std::vector<Model>& smtrat::Module::allModels ( ) const [inline], [inherited]`

Returns

All satisfying assignments, if existent.

0.15.229.4.13 anAnswerFound() `bool smtrat::Module::anAnswerFound ( ) const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.229.4.14 answerFound() `const smtrat::Conditionals& smtrat::Module::answerFound ( ) const [inline], [inherited]`

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.229.4.15 appliedPreprocessing() `bool smtrat::PModule::appliedPreprocessing ( ) const [inline], [inherited]`

Returns

true, if the current received formula has been simplified and the result of this simplification is stored in the passed formula.

0.15.229.4.16 backendsModel() `const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.229.4.17 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline],
[protected], [inherited]
```

0.15.229.4.18 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.229.4.19 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline],
[protected], [inherited]
```

0.15.229.4.20 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.229.4.21 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.

0.15.229.4.22 checkCore()

```
template<typename Settings >
Answer smtrat::ICEModule< Settings >::checkCore ( ) [virtual]
```

Checks the received formula for consistency.

Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.229.4.23 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.

0.15.229.4.24 checkModel()

```
unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]
```

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.229.4.25 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.229.4.26 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

0.15.229.4.27 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

0.15.229.4.28 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

0.15.229.4.29 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.229.4.30 collectOrigins() [1/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

0.15.229.4.31 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.

0.15.229.4.32 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

0.15.229.4.33 constraintsToInform() `const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns

The constraints which the backends must be informed about.

0.15.229.4.34 currentlySatisfied()

`virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.229.4.35 currentlySatisfiedByBackend()

`unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.229.4.36 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.

0.15.229.4.37 deinformCore()

`virtual void smtrat::Module::deinformCore ( [[maybe_unused] ] 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.

0.15.229.4.38 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

0.15.229.4.39 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 >.

0.15.229.4.40 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

0.15.229.4.41 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
            const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

0.15.229.4.42 firstSubformulaToPass()

```
ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]
```

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.229.4.43 firstUncheckedReceivedSubformula()

```
ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]
```

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.229.4.44 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.229.4.45 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasible↩ Subset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.229.4.46 getBackendsAllModels()

`void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.229.4.47 getBackendsModel()

`void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.229.4.48 getInfeasibleSubsets() [1/2]

`void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.229.4.49 getInfeasibleSubsets() [2/2]

`std::vector< FormulaSetT > smtrat::Module::get↩ InfeasibleSubsets ( 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.

0.15.229.4.50 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module↩ ::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.229.4.51 getModelEqualities()

`std::list< std::vector< carl::Variable > > smtrat::Module↩ ::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.229.4.52 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
const FormulaT & _formula,
FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.229.4.53 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
const FormulaT & _formula,
FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.229.4.54 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.

0.15.229.4.55 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.

0.15.229.4.56 hasLemmas()

```
bool smtrat::Module::hasLemmas ( ) [inline], [inherited]
```

Checks whether this module or any of its backends provides any lemmas.

0.15.229.4.57 hasValidInfeasibleSubset()

```
bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]
```

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.229.4.58 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.229.4.59 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasible↩ Subsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.229.4.60 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.

0.15.229.4.61 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.

0.15.229.4.62 informCore() `virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.229.4.63 informedConstraints()

`const carl::FastSet<FormulaT>& smtrat::Module::informedConstraints ( ) const [inline], [inherited]`

Returns

The position of the first constraint of which no backend has been informed about.

0.15.229.4.64 init()

`void smtrat::Module::init ( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not be called twice and only before the first runBackends call.
Reimplemented in smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntBlastModule< Settings >, smtrat::FPPModule< Settings >, smtrat::CurryModule< Settings >, and smtrat::BVModule< Settings >.

0.15.229.4.65 is_minimizing()

`bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.229.4.66 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.

0.15.229.4.67 isPreprocessor()

`bool smtrat::PModule::isPreprocessor ( ) const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.229.4.68 lemmas()

`const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.229.4.69 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.

0.15.229.4.70 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.229.4.71 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.

0.15.229.4.72 moduleName() `virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]`

Returns

The name of the given module type as name.

Reimplemented in smtrat::VSModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::STropModule< Settings >, smtrat::SplitSOSModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntEqModule< Settings >, smtrat::IntBlastModule< Settings >, smtrat::IncWidthModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::GBModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::CurryModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::BVModule< Settings >, and smtrat::BEModule< Settings >.

0.15.229.4.73 objective() `carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

0.15.229.4.74 originInReceivedFormula() `bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.229.4.75 passedFormulaBegin() `ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.229.4.76 passedFormulaEnd() `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.229.4.77 pPassedFormula() `const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.229.4.78 pReceivedFormula() `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.229.4.79 print() `void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.229.4.80 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.

0.15.229.4.81 printInfeasibleSubsets() `void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.229.4.82 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.

0.15.229.4.83 printPassedFormula()

`void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.229.4.84 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.229.4.85 probablyLooping()

`bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.229.4.86 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.229.4.87 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

0.15.229.4.88 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const [inline], [inherited]
```

0.15.229.4.89 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.

0.15.229.4.90 removeCore() [1/2]

```
virtual void smtrat::Module::removeCore (
    [[maybe_unused] ] 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.

0.15.229.4.91 removeCore() [2/2]

```
template<typename Settings >
void smtrat::ICEModule< Settings >::removeCore (
    ModuleInput::const_iterator _subformula )
```

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.

0.15.229.4.92 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

0.15.229.4.93 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

0.15.229.4.94 rPassedFormula() `const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]`

Returns

A reference to the formula passed to the backends of this module.

0.15.229.4.95 rReceivedFormula() `const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]`

Returns

A reference to the formula passed to this module.

0.15.229.4.96 runBackends() [1/2] `virtual Answer smtrat::PModule::runBackends ( ) [inline], [virtual], [inherited]`

Reimplemented from smtrat::Module.

0.15.229.4.97 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.

0.15.229.4.98 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.

0.15.229.4.99 setThreadPriority()

```
void smtrat::Module::setThreadPriority (
    thread_priority _threadPriority ) [inline], [inherited]
```

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.229.4.100 solverState() `Answer smtrat::Module::solverState ( ) const [inline], [inherited]`

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.229.4.101 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 !=.

0.15.229.4.102 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.229.4.103 updateAllModels() `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.229.4.104 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.229.4.105 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.

0.15.229.4.106 usedBackends() `const std::vector<Module*>& smtrat::Module::usedBackends ( ) const [inline], [inherited]`

Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.229.5 Field Documentation

0.15.229.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.229.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.229.5.3 mBackendsFoundAnswer `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.229.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.229.5.5 mFinalCheck `bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.229.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.229.5.7 mFirstSubformulaToPass `ModuleInput::iterator smtrat::Module::mFirstSubformulaTo↩ Pass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.229.5.8 mFirstUncheckedReceivedSubformula `ModuleInput::const_iterator smtrat::Module↩ ::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.229.5.9 mFoundAnswer `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.229.5.10 mFullCheck `bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.229.5.11 mInfeasibleSubsets `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.229.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.229.5.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.229.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.229.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.229.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.229.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.229.5.18 mObjectiveVariable

`carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.229.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]`

Reusable splitting variables.

0.15.229.5.20 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.229.5.21 mSmallerMusesCheckCounter

`unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.229.5.22 mSolverState

`std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.229.5.23 mTheoryPropagations

`std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]`

0.15.229.5.24 mUsedBackends `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.229.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.230 smtrat::ICPModule< Settings > Class Template Reference

`#include <ICPModule.h>`

Data Structures

- struct comp

  Typedefs:
- struct formulaPtrComp
- struct linearVariable
- struct weights

Public Types

- typedef carl::FastPointerMap< Poly *, weights > WeightMap
- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- ICPModule (const ModuleInput *, Conditionals &, Manager *const =NULL)

  Constructors:
- ~ICPModule ()

  Destructor:
- bool informCore (const FormulaT &)
- bool addCore (ModuleInput::const_iterator)
- void removeCore (ModuleInput::const_iterator)
- 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 bool informCore ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] 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 ()
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const

- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.230.1 Member Typedef Documentation

0.15.230.1.1 SettingsType `template<class Settings >`

```
typedef Settings smtrat::ICPModule< Settings >::SettingsType
```

0.15.230.1.2 WeightMap `template<class Settings >`

```
typedef carl::FastPointerMap<Poly*, weights> smtrat::ICPModule< Settings >::WeightMap
```

0.15.230.2 Member Enumeration Documentation

0.15.230.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

0.15.230.3 Constructor & Destructor Documentation

0.15.230.3.1 ICPModule() `template<class Settings >`

```
smtrat::ICPModule< Settings >::ICPModule (
            const ModuleInput * ,
```

```
            Conditionals & ,
            Manager * const  = NULL )
```

Constructors:

0.15.230.3.2 ~ICPModule()

```
template<class Settings >
smtrat::ICPModule< Settings >::~ICPModule ( )
```

Destructor:

0.15.230.4 Member Function Documentation

0.15.230.4.1 add()

```
bool smtrat::Module::add (
            ModuleInput::const_iterator _subformula )  [inherited]
```

The module has to take the given sub-formula of the received formula into account.

Parameters

_subformula	The sub-formula to take additionally into account.

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

0.15.230.4.2 addConstraintToInform()

```
void smtrat::Module::addConstraintToInform (
            const FormulaT & _constraint )  [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

0.15.230.4.3 addCore() [1/2]

```
virtual bool smtrat::Module::addCore (
            [[maybe_unused] ] 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.

0.15.230.4.4 addCore() [2/2]

```
template<class Settings >
bool smtrat::ICPModule< Settings >::addCore (
            ModuleInput::const_iterator  )
```

0.15.230.4.5 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.230.4.6 addLemma()

```
void smtrat::Module::addLemma (
    const FormulaT & _lemma,
    const LemmaType & _lt = LemmaType::NORMAL,
    const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline], [inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.230.4.7 addOrigin()

```
void smtrat::Module::addOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.230.4.8 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::addReceivedSubformulaToPassedFormula (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.230.4.9 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
```

```
const FormulaT & _formula )  [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.230.4.10 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula,
            const FormulaT & _origin )  [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.230.4.11 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.

0.15.230.4.12 allModels() `const std::vector<Model>& smtrat::Module::allModels ( ) const [inline], [inherited]`

Returns

All satisfying assignments, if existent.

0.15.230.4.13 anAnswerFound() `bool smtrat::Module::anAnswerFound ( ) const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.230.4.14 answerFound() `const smtrat::Conditionals& smtrat::Module::answerFound ( ) const [inline], [inherited]`

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.230.4.15 backendsModel() `const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.230.4.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.230.4.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.230.4.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.230.4.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.230.4.20 check()

```
Answer smtrat::Module::check (
    bool _final = false,
    bool _full = true,
    carl::Variable _objective = carl::Variable::NO_VARIABLE ) [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.230.4.21 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.

0.15.230.4.22 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.

0.15.230.4.23 checkModel()

```
unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]
```

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.230.4.24 cleanModel()

```
void smtrat::Module::cleanModel ( ) const [inline], [protected],
[inherited]
```

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.230.4.25 clearLemmas()

```
void smtrat::Module::clearLemmas ( ) [inline], [inherited]
```

Deletes all yet found lemmas.

0.15.230.4.26 clearModel()

```
void smtrat::Module::clearModel ( ) const [inline], [protected],
[inherited]
```

Clears the assignment, if any was found.

0.15.230.4.27 clearModels()

```
void smtrat::Module::clearModels ( ) const [inline], [protected],
[inherited]
```

Clears all assignments, if any was found.

0.15.230.4.28 clearPassedFormula()

`void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.230.4.29 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

0.15.230.4.30 collectOrigins() [2/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.
If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.230.4.31 collectTheoryPropagations()

`void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

0.15.230.4.32 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ToInform ( ) const [inline], [inherited]`

Returns

The constraints which the backends must be informed about.

0.15.230.4.33 currentlySatisfied()

```
virtual unsigned smtrat::Module::currentlySatisfied (
    const FormulaT & ) const [inline], [virtual], [inherited]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.230.4.34 currentlySatisfiedByBackend()

```
unsigned smtrat::Module::currentlySatisfiedByBackend (
    const FormulaT & _formula ) const [inherited]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.230.4.35 deinform()

```
void smtrat::Module::deinform (
    const FormulaT & _constraint ) [inherited]
```

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.230.4.36 deinformCore()

```
virtual void smtrat::Module::deinformCore (
    [[maybe_unused] ] 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.

0.15.230.4.37 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

0.15.230.4.38 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.

0.15.230.4.39 excludeNotReceivedVariablesFromModel()

`void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

0.15.230.4.40 findBestOrigin()

`std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin ( const std::vector< FormulaT > & _origins ) const [protected], [inherited]`

Parameters

_origins	

Returns

0.15.230.4.41 firstSubformulaToPass()

`ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.230.4.42 firstUncheckedReceivedSubformula()

`ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.230.4.43 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.230.4.44 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.230.4.45 getBackendsAllModels()

`void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.230.4.46 getBackendsModel()

`void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.230.4.47 getCurrentBoxAsFormulas()

`template<class Settings > FormulasT smtrat::ICPModule< Settings >::getCurrentBoxAsFormulas ( ) const`

0.15.230.4.48 getCurrentBoxAsIntervals() template<class Settings >

EvalRationalIntervalMap smtrat::ICPModule< Settings >::getCurrentBoxAsIntervals ( ) const

0.15.230.4.49 getInfeasibleSubsets() [1/2] void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]

Copies the infeasible subsets of the passed formula.

0.15.230.4.50 getInfeasibleSubsets() [2/2] std::vector< FormulaSetT > smtrat::Module::get↩InfeasibleSubsets (

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.

0.15.230.4.51 getInformationRelevantFormulas() const std::set< FormulaT > & smtrat::Module↩::getInformationRelevantFormulas ( ) [protected], [inherited]

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.230.4.52 getModelEqualities() std::list< std::vector< carl::Variable > > smtrat::Module↩::getModelEqualities ( ) const [virtual], [inherited]

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.230.4.53 getOrigins() [1/3] void smtrat::Module::getOrigins (

const FormulaT & *_formula*,
FormulaSetT & *_origins*) const [inline], [protected], [inherited]

Parameters

_formula	
_origins	

0.15.230.4.54 getOrigins() [2/3]

`void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.230.4.55 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.

0.15.230.4.56 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.

0.15.230.4.57 hasLemmas()

`bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

0.15.230.4.58 hasValidInfeasibleSubset()

`bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]`

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.230.4.59 id()

`std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.230.4.60 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.

0.15.230.4.61 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.

0.15.230.4.62 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.

0.15.230.4.63 informCore() [1/2] `virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.230.4.64 informCore() [2/2] `template<class Settings > bool smtrat::ICPModule< Settings >::informCore ( const FormulaT & )`

0.15.230.4.65 informedConstraints() `const carl::FastSet<FormulaT>& smtrat::Module::informedConstraints ( ) const [inline], [inherited]`

Returns

The position of the first constraint of which no backend has been informed about.

0.15.230.4.66 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::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntBlastModule< Settings >, smtrat::FPPModule< Settings >, smtrat::CurryModule< Settings >, and smtrat::BVModule< Settings >.

0.15.230.4.67 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.230.4.68 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.

0.15.230.4.69 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().

0.15.230.4.70 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.

0.15.230.4.71 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.

0.15.230.4.72 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.230.4.73 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.

0.15.230.4.74 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.

0.15.230.4.75 objective() `carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

0.15.230.4.76 originInReceivedFormula() `bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.230.4.77 passedFormulaBegin() `ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.230.4.78 passedFormulaEnd() `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.230.4.79 pPassedFormula() `const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.230.4.80 pReceivedFormula() `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.230.4.81 print() `void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.230.4.82 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.

0.15.230.4.83 printInfeasibleSubsets() `void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.230.4.84 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.

0.15.230.4.85 printPassedFormula()

`void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.230.4.86 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.230.4.87 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.

0.15.230.4.88 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.230.4.89 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

0.15.230.4.90 receivedVariable()

`bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

0.15.230.4.91 remove()

`void smtrat::Module::remove ( ModuleInput::const_iterator _subformula ) [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.
However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

Parameters

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

0.15.230.4.92 removeCore() [1/2]

`virtual void smtrat::Module::removeCore ( [[maybe_unused] ] 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.

0.15.230.4.93 removeCore() [2/2]

`template<class Settings > void smtrat::ICPModule< Settings >::removeCore ( ModuleInput::const_iterator )`

0.15.230.4.94 removeOrigin()

`std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin ( ModuleInput::iterator _formula, const FormulaT & _origin ) [inline], [protected], [inherited]`

0.15.230.4.95 removeOrigins()

`std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins ( ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]`

0.15.230.4.96 rPassedFormula()

`const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]`

Returns

A reference to the formula passed to the backends of this module.

0.15.230.4.97 rReceivedFormula()

`const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]`

Returns

A reference to the formula passed to this module.

0.15.230.4.98 runBackends() [1/2]

`virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]`

Reimplemented in smtrat::PModule.

0.15.230.4.99 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.

0.15.230.4.100 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.

0.15.230.4.101 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.

0.15.230.4.102 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.

0.15.230.4.103 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 !=.

0.15.230.4.104 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.230.4.105 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 >.

0.15.230.4.106 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.230.4.107 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.

0.15.230.4.108 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).

0.15.230.5 Field Documentation

0.15.230.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.230.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.230.5.3 mBackendsFoundAnswer

`std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.230.5.4 mConstraintsToInform

`carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.230.5.5 mFinalCheck

`bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.230.5.6 mFirstPosInLastBranches

`std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.230.5.7 mFirstSubformulaToPass

`ModuleInput::iterator smtrat::Module::mFirstSubformulaTo↩Pass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.230.5.8 mFirstUncheckedReceivedSubformula

`ModuleInput::const_iterator smtrat::Module↩::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.230.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.230.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.230.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.230.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.230.5.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std↩::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.230.5.14 mLemmas `std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.230.5.15 mModel `Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.230.5.16 mModelComputed `bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.230.5.17 mNumOfBranchVarsToStore `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.230.5.18 mObjectiveVariable `carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.230.5.19 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

0.15.230.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.230.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.230.5.22 mSolverState `std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.230.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

0.15.230.5.24 mUsedBackends `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.230.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.231 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 is_linear ()
- EvalDoubleIntervalMap::const_iterator intervalPosition () const
- const DoubleInterval & interval () const
- void setInterval (const DoubleInterval &_interval)
- void setUnmodified ()
- void setExternalUnmodified ()
- void setExternalModified ()
- void setInternalUnmodified ()
- Updated isInternalUpdated () const
- Updated isExternalUpdated () const
- const smtrat::FormulaT & internalLeftBound () const
- const smtrat::FormulaT & internalRightBound () const
- ModuleInput::iterator externalLeftBound () const
- ModuleInput::iterator externalRightBound () const
- void setInternalLeftBound (const smtrat::FormulaT &_left)
- void setInternalRightBound (const smtrat::FormulaT &_right)
- void setExternalLeftBound (ModuleInput::iterator _left)
- void setExternalRightBound (ModuleInput::iterator _right)
- Updated isInternalBoundsSet () const
- bool isOriginal () const
- bool operator< (IcpVariable const &rhs) const

Friends

- std::ostream & operator<< (std::ostream &os, const IcpVariable &_var)

0.15.231.1 Constructor & Destructor Documentation

0.15.231.1.1 IcpVariable()

```
smtrat::icp::IcpVariable::IcpVariable (
    carl::Variable::Arg _var,
    bool _original,
    ModuleInput::iterator _defaultPosition,
    EvalDoubleIntervalMap::iterator _intervalPos,
    const LRAVariable * _lraVar = NULL ) [inline]
```

0.15.231.1.2 ~IcpVariable() smtrat::icp::IcpVariable::~IcpVariable ( ) [inline]

0.15.231.2 Member Function Documentation

0.15.231.2.1 addCandidate() void smtrat::icp::IcpVariable::addCandidate (ContractionCandidate * *_candidate*) [inline]

0.15.231.2.2 addCandidates() void smtrat::icp::IcpVariable::addCandidates (const ContractionCandidates & *_candidates*) [inline]

0.15.231.2.3 addOriginalConstraint() void smtrat::icp::IcpVariable::addOriginalConstraint (const FormulaT & *_constraint*) [inline]

0.15.231.2.4 candidates() ContractionCandidates& smtrat::icp::IcpVariable::candidates ( ) [inline]

0.15.231.2.5 decrementActivity() void smtrat::icp::IcpVariable::decrementActivity ( ) [inline]

0.15.231.2.6 externalLeftBound() ModuleInput::iterator smtrat::icp::IcpVariable::externalLeft↩ Bound ( ) const [inline]

0.15.231.2.7 externalRightBound() ModuleInput::iterator smtrat::icp::IcpVariable::external↩ RightBound ( ) const [inline]

0.15.231.2.8 incrementActivity() void smtrat::icp::IcpVariable::incrementActivity ( ) [inline]

0.15.231.2.9 internalLeftBound() const smtrat::FormulaT& smtrat::icp::IcpVariable::internal↩ LeftBound ( ) const [inline]

0.15.231.2.10 internalRightBound() const smtrat::FormulaT& smtrat::icp::IcpVariable::internal↩ RightBound ( ) const [inline]

0.15.231.2.11 interval() const DoubleInterval& smtrat::icp::IcpVariable::interval ( ) const [inline]

0.15.231.2.12 intervalPosition() EvalDoubleIntervalMap::const_iterator smtrat::icp::IcpVariable↩ ::intervalPosition ( ) const [inline]

0.15.231.2.13 is_linear() bool smtrat::icp::IcpVariable::is_linear ( ) [inline]

0.15.231.2.14 isActive()

`bool smtrat::icp::IcpVariable::isActive ( ) const [inline]`

0.15.231.2.15 isExternalUpdated()

`Updated smtrat::icp::IcpVariable::isExternalUpdated ( ) const [inline]`

0.15.231.2.16 isInternalBoundsSet()

`Updated smtrat::icp::IcpVariable::isInternalBoundsSet ( ) const [inline]`

0.15.231.2.17 isInternalUpdated()

`Updated smtrat::icp::IcpVariable::isInternalUpdated ( ) const [inline]`

0.15.231.2.18 isOriginal()

`bool smtrat::icp::IcpVariable::isOriginal ( ) const [inline]`

0.15.231.2.19 lraVar()

`const LRAVariable* smtrat::icp::IcpVariable::lraVar ( ) const [inline]`

0.15.231.2.20 operator<()

```
bool smtrat::icp::IcpVariable::operator< (
IcpVariable const & rhs ) const [inline]
```

0.15.231.2.21 print()

```
void smtrat::icp::IcpVariable::print (
std::ostream & _out = std::cout,
bool _withContractionCandidates = false ) const [inline]
```

0.15.231.2.22 removeOriginalConstraint()

```
void smtrat::icp::IcpVariable::removeOriginalConstraint (
const FormulaT & _constraint ) [inline]
```

0.15.231.2.23 setExternalLeftBound()

```
void smtrat::icp::IcpVariable::setExternalLeftBound (
ModuleInput::iterator _left ) [inline]
```

0.15.231.2.24 setExternalModified()

`void smtrat::icp::IcpVariable::setExternalModified ( ) [inline]`

0.15.231.2.25 setExternalRightBound()

```
void smtrat::icp::IcpVariable::setExternalRightBound (
ModuleInput::iterator _right ) [inline]
```

0.15.231.2.26 setExternalUnmodified()

`void smtrat::icp::IcpVariable::setExternalUnmodified ( ) [inline]`

0.15.231.2.27 setInternalLeftBound()

```
void smtrat::icp::IcpVariable::setInternalLeftBound (
const smtrat::FormulaT & _left ) [inline]
```

0.15.231.2.28 setInternalRightBound()

```
void smtrat::icp::IcpVariable::setInternalRightBound ( const smtrat::FormulaT & _right ) [inline]
```

0.15.231.2.29 setInternalUnmodified()

```
void smtrat::icp::IcpVariable::setInternalUnmodified ( ) [inline]
```

0.15.231.2.30 setInterval()

```
void smtrat::icp::IcpVariable::setInterval ( const DoubleInterval & _interval ) [inline]
```

0.15.231.2.31 setLraVar()

```
void smtrat::icp::IcpVariable::setLraVar ( const LRAVariable * _lraVar ) [inline]
```

0.15.231.2.32 setUnmodified()

```
void smtrat::icp::IcpVariable::setUnmodified ( ) [inline]
```

0.15.231.2.33 var()

```
carl::Variable::Arg smtrat::icp::IcpVariable::var ( ) const [inline]
```

0.15.231.3 Friends And Related Function Documentation

0.15.231.3.1 operator<<

```
std::ostream& operator<< ( std::ostream & os, const IcpVariable & _var ) [friend]
```

0.15.232 smtrat::icp::icpVariableComp Struct Reference

```
#include <IcpVariable.h>
```

Public Member Functions

- bool operator() (const IcpVariable *const _lhs, const IcpVariable *const _rhs) const

0.15.232.1 Member Function Documentation

0.15.232.1.1 operator()()

```
bool smtrat::icp::icpVariableComp::operator() ( const IcpVariable *const _lhs, const IcpVariable *const _rhs ) const [inline]
```

0.15.233 smtrat::parser::Identifier Struct Reference

```
#include <Common.h>
```

Public Member Functions

- Identifier ()
- Identifier (const std::string &symbol)
- Identifier (const std::string &symbol, const std::vector< std::size_t > &indices)
- Identifier (const std::string &symbol, const std::vector< Integer > &indices)
- Identifier & operator= (const Identifier &i)
- Identifier (const Identifier &i)

- ~Identifier ()
- operator std::string () const

Data Fields

- std::string symbol
- std::vector< std::size_t > * indices

0.15.233.1 Constructor & Destructor Documentation

0.15.233.1.1 Identifier() [1/5] `smtrat::parser::Identifier::Identifier ( ) [inline]`

0.15.233.1.2 Identifier() [2/5] `smtrat::parser::Identifier::Identifier ( const std::string & symbol ) [inline]`

0.15.233.1.3 Identifier() [3/5] `smtrat::parser::Identifier::Identifier ( const std::string & symbol, const std::vector< std::size_t > & indices ) [inline]`

0.15.233.1.4 Identifier() [4/5] `smtrat::parser::Identifier::Identifier ( const std::string & symbol, const std::vector< Integer > & indices ) [inline]`

0.15.233.1.5 Identifier() [5/5] `smtrat::parser::Identifier::Identifier ( const Identifier & i ) [inline]`

0.15.233.1.6 ~Identifier() `smtrat::parser::Identifier::~Identifier ( ) [inline]`

0.15.233.2 Member Function Documentation

0.15.233.2.1 operator std::string() `smtrat::parser::Identifier::operator std::string ( ) const [inline]`

0.15.233.2.2 operator=() `Identifier& smtrat::parser::Identifier::operator= ( const Identifier & i ) [inline]`

0.15.233.3 Field Documentation

0.15.233.3.1 indices `std::vector<std::size_t>* smtrat::parser::Identifier::indices`

0.15.233.3.2 symbol `std::string smtrat::parser::Identifier::symbol`

0.15.234 smtrat::parser::IdentifierParser Struct Reference

`#include <Identifier.h>`

Public Member Functions

- IdentifierParser ()

Data Fields

- SymbolParser symbol
- NumeralParser numeral
- qi::rule< Iterator, Identifier(), Skipper > main
- qi::rule< Iterator, Identifier(), Skipper > indexed

0.15.234.1 Constructor & Destructor Documentation

0.15.234.1.1 IdentifierParser() `smtrat::parser::IdentifierParser::IdentifierParser ( ) [inline]`

0.15.234.2 Field Documentation

0.15.234.2.1 indexed `qi::rule<Iterator, Identifier(), Skipper> smtrat::parser::IdentifierParser::indexed`

0.15.234.2.2 main `qi::rule<Iterator, Identifier(), Skipper> smtrat::parser::IdentifierParser::main`

0.15.234.2.3 numeral `NumeralParser smtrat::parser::IdentifierParser::numeral`

0.15.234.2.4 symbol `SymbolParser smtrat::parser::IdentifierParser::symbol`

0.15.235 smtrat::cad::debug::IDSanitizer Struct Reference

`#include <TikzHistoryPrinter.h>`

Public Member Functions

- IDSanitizer (const std::string &prefix)
- std::string operator() (const std::string &raw)

0.15.235.1 Constructor & Destructor Documentation

0.15.235.1.1 IDSanitizer() `smtrat::cad::debug::IDSanitizer::IDSanitizer ( const std::string & prefix ) [inline]`

0.15.235.2 Member Function Documentation

0.15.235.2.1 operator()() `std::string smtrat::cad::debug::IDSanitizer::operator() ( const std::string & raw ) [inline]`

0.15.236 smtrat::covering_ng::formula::node_ds::IFF Struct Reference

`#include <FormulaEvaluation.h>`

Data Fields

- std::vector< Node > subformulas

0.15.236.1 Field Documentation

0.15.236.1.1 subformulas `std::vector<Node> smtrat::covering_ng::formula::node_ds::IFF::subformulas`

0.15.237 smtrat::mcsat::fm::IgnoreCoreSettings Struct Reference

`#include <Explanation.h>`

Static Public Attributes

- static constexpr bool use_all_constraints = true

0.15.237.1 Field Documentation

0.15.237.1.1 use_all_constraints `constexpr bool smtrat::mcsat::fm::IgnoreCoreSettings::use_all↩ _constraints = true [static], [constexpr]`

0.15.238 smtrat::IncWidthModule< Settings > Class Template Reference

`#include <IncWidthModule.h>`

Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- IncWidthModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=NULL)
- ~IncWidthModule ()
- bool addCore (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- void removeCore (ModuleInput::const_iterator _subformula)

  Removes the subformula of the received formula at the given position to the considered ones of this module.
- void updateModel () const

  Updates the current assignment into the model.
- Answer checkCore ()

  Checks the received formula for consistency.
- bool inform (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void 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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.
- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)
- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

  Copies the infeasible subsets of the passed formula.
- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.

- virtual Answer runBackends ()
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.238.1 Member Typedef Documentation

0.15.238.1.1 SettingsType `template<typename Settings >`

```
typedef Settings smtrat::IncWidthModule< Settings >::SettingsType
```

0.15.238.2 Member Enumeration Documentation

0.15.238.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

0.15.238.3 Constructor & Destructor Documentation

0.15.238.3.1 IncWidthModule() template<typename Settings >

```
smtrat::IncWidthModule< Settings >::IncWidthModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = NULL )
```

0.15.238.3.2 ~IncWidthModule() template<typename Settings >

```
smtrat::IncWidthModule< Settings >::~IncWidthModule ( )
```

0.15.238.4 Member Function Documentation

0.15.238.4.1 add() bool smtrat::Module::add (

```
            ModuleInput::const_iterator _subformula )  [inherited]
```

The module has to take the given sub-formula of the received formula into account.

Parameters

_subformula	The sub-formula to take additionally into account.

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

0.15.238.4.2 addConstraintToInform() void smtrat::Module::addConstraintToInform (

```
            const FormulaT & _constraint )  [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

0.15.238.4.3 addCore() [1/2] virtual bool smtrat::Module::addCore (

```
            [[maybe_unused] ] 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.

0.15.238.4.4 addCore() [2/2] template<typename Settings >

```
bool smtrat::IncWidthModule< Settings >::addCore (
            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.

0.15.238.4.5 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula
(
            const FormulaT & formula )  [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.238.4.6 addLemma()

```
void smtrat::Module::addLemma (
            const FormulaT & _lemma,
            const LemmaType & _lt = LemmaType::NORMAL,
            const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) )  [inline],
[inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.238.4.7 addOrigin()

```
void smtrat::Module::addOrigin (
            ModuleInput::iterator _formula,
            const FormulaT & _origin )  [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.238.4.8 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
            ModuleInput::const_iterator _subformula )  [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.238.4.9 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula )  [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.238.4.10 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula,
            const FormulaT & _origin )  [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.238.4.11 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.

0.15.238.4.12 allModels() `const std::vector<Model>& smtrat::Module::allModels ( ) const [inline], [inherited]`

Returns

All satisfying assignments, if existent.

0.15.238.4.13 anAnswerFound() `bool smtrat::Module::anAnswerFound ( ) const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.238.4.14 answerFound() `const smtrat::Conditionals& smtrat::Module::answerFound ( ) const [inline], [inherited]`

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.238.4.15 backendsModel() `const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.238.4.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.238.4.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.238.4.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline],
[protected], [inherited]
```

0.15.238.4.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.238.4.20 check()

```
Answer smtrat::Module::check (
    bool _final = false,
    bool _full = true,
    carl::Variable _objective = carl::Variable::NO_VARIABLE ) [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.238.4.21 checkCore()

`template<typename 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.

0.15.238.4.22 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.

0.15.238.4.23 checkModel()

`unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]`

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.238.4.24 cleanModel()

`void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.238.4.25 clearLemmas()

`void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

0.15.238.4.26 clearModel()

`void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

0.15.238.4.27 clearModels()

`void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

0.15.238.4.28 clearPassedFormula()

`void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.238.4.29 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

0.15.238.4.30 collectOrigins() [2/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.
If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.238.4.31 collectTheoryPropagations()

`void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

0.15.238.4.32 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns

The constraints which the backends must be informed about.

0.15.238.4.33 currentlySatisfied()

```
virtual unsigned smtrat::Module::currentlySatisfied (
    const FormulaT & ) const [inline], [virtual], [inherited]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.238.4.34 currentlySatisfiedByBackend()

```
unsigned smtrat::Module::currentlySatisfiedByBackend (
    const FormulaT & _formula ) const [inherited]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.238.4.35 deinform()

```
void smtrat::Module::deinform (
    const FormulaT & _constraint ) [inherited]
```

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.238.4.36 deinformCore()

```
virtual void smtrat::Module::deinformCore (
    [[maybe_unused] ] 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.

0.15.238.4.37 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

0.15.238.4.38 eraseSubformulaFromPassedFormula()

```
ModuleInput::iterator smtrat::Module::erase↩
SubformulaFromPassedFormula (
    ModuleInput::iterator _subformula,
    bool _ignoreOrigins = false ) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_subformula	The sub-formula to remove from the passed formula.

Parameters

_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 >.

0.15.238.4.39 excludeNotReceivedVariablesFromModel()

`void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

0.15.238.4.40 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
            const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

0.15.238.4.41 firstSubformulaToPass()

`ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.238.4.42 firstUncheckedReceivedSubformula()

`ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.238.4.43 freeSplittingVariable()

```
static void smtrat::Module::freeSplittingVariable (
            const FormulaT & _splittingVariable ) [inline], [static], [inherited]
```

0.15.238.4.44 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.238.4.45 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.238.4.46 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.238.4.47 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.238.4.48 getInfeasibleSubsets() [2/2] `std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets ( const Module & _backend ) const [protected], [inherited]`

Get the infeasible subsets the given backend provides.
Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

Parameters

_backend	The backend from which to obtain the infeasible subsets.

Returns

The infeasible subsets the given backend provides.

0.15.238.4.49 getInformationRelevantFormulas() `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.238.4.50 getModelEqualities() `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.238.4.51 getOrigins() [1/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.238.4.52 getOrigins() [2/3]

`void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.238.4.53 getOrigins() [3/3]

`const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]`

Gets the origins of the passed formula at the given position.

Parameters

_formula	The position of a formula in the passed formulas.

Returns

The origins of the passed formula at the given position.

0.15.238.4.54 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.

0.15.238.4.55 hasLemmas()

`bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

0.15.238.4.56 hasValidInfeasibleSubset()

`bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]`

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.238.4.57 id()

`std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.238.4.58 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasible↩ Subsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.238.4.59 inform() `bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.
It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

Parameters

_constraint	The constraint to inform about.

Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

0.15.238.4.60 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.

0.15.238.4.61 informCore() `virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.238.4.62 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.

0.15.238.4.63 init() `void smtrat::Module::init ( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not be called twice and only before the first runBackends call.
Reimplemented in smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntBlastModule< Settings >, smtrat::FPPModule< Settings >, smtrat::CurryModule< Settings >, and smtrat::BVModule< Settings >.

0.15.238.4.64 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.238.4.65 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.

0.15.238.4.66 isPreprocessor() `bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.238.4.67 lemmas() `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.238.4.68 merge() `std::vector< FormulaT > smtrat::Module::merge ( const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.
({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

Parameters

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns

The vector being the two given vectors merged.

0.15.238.4.69 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.238.4.70 modelsDisjoint()

`bool smtrat::Module::modelsDisjoint ( const Model & _modelA, const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

Parameters

_modelA	The first model to check for.
_modelB	The second model to check for.

Returns

true, if there is no variable assigned by both models; false, otherwise.

0.15.238.4.71 moduleName()

`template<typename Settings > std::string smtrat::IncWidthModule< Settings >::moduleName ( ) const [inline], [virtual]`

Returns

The name of the given module type as name.

Reimplemented from smtrat::Module.

0.15.238.4.72 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

0.15.238.4.73 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.238.4.74 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.238.4.75 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.238.4.76 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.238.4.77 pReceivedFormula() `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.238.4.78 print() `void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.238.4.79 printAllModels() `void smtrat::Module::printAllModels ( std::ostream & _out = std::cout ) [inherited]`

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.238.4.80 printInfeasibleSubsets() `void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.238.4.81 printModel() `void smtrat::Module::printModel ( std::ostream & _out = std::cout ) const [inherited]`

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.238.4.82 printPassedFormula() `void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.238.4.83 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.238.4.84 probablyLooping()

`bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.238.4.85 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.238.4.86 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

0.15.238.4.87 receivedVariable()

`bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

0.15.238.4.88 remove()

`void smtrat::Module::remove ( ModuleInput::const_iterator _subformula ) [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.
However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

Parameters

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

0.15.238.4.89 removeCore() [1/2]

```
virtual void smtrat::Module::removeCore (
    [[maybe_unused] ] 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.

0.15.238.4.90 removeCore() [2/2]

```
template<typename Settings >
void smtrat::IncWidthModule< Settings >::removeCore (
    ModuleInput::const_iterator _subformula )
```

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.

0.15.238.4.91 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

0.15.238.4.92 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

0.15.238.4.93 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.238.4.94 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to this module.

0.15.238.4.95 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( ) [inline],
[protected], [virtual], [inherited]
```

Reimplemented in smtrat::PModule.

0.15.238.4.96 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.

0.15.238.4.97 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.

0.15.238.4.98 setThreadPriority()

```
void smtrat::Module::setThreadPriority (
    thread_priority _threadPriority ) [inline], [inherited]
```

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.238.4.99 solverState()

```
Answer smtrat::Module::solverState ( ) const [inline], [inherited]
```

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.238.4.100 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 !=.

0.15.238.4.101 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.238.4.102 updateAllModels() `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.238.4.103 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.238.4.104 updateModel() `template<typename Settings > void smtrat::IncWidthModule< Settings >::updateModel ( ) const [virtual]`

Updates the current assignment into the model.
Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.
Reimplemented from smtrat::Module.

0.15.238.4.105 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).

0.15.238.5 Field Documentation

0.15.238.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.238.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.238.5.3 mBackendsFoundAnswer `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.238.5.4 mConstraintsToInform

`carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.238.5.5 mFinalCheck

`bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.238.5.6 mFirstPosInLastBranches

`std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.238.5.7 mFirstSubformulaToPass

`ModuleInput::iterator smtrat::Module::mFirstSubformulaTo↩ Pass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.238.5.8 mFirstUncheckedReceivedSubformula

`ModuleInput::const_iterator smtrat::Module↩ ::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.238.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.238.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.238.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.238.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.238.5.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std↩ ::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.238.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.238.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.238.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.238.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.238.5.18 mObjectiveVariable

`carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.238.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

0.15.238.5.20 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.238.5.21 mSmallerMusesCheckCounter

`unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.238.5.22 mSolverState

`std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.238.5.23 mTheoryPropagations

`std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

0.15.238.5.24 mUsedBackends

`std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.238.5.25 mVariableCounters

`std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.239 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

0.15.239.1 Constructor & Destructor Documentation

0.15.239.1.1 ~IndexedFunctionInstantiator()

```
virtual smtrat::parser::IndexedFunctionInstantiator↩
::~IndexedFunctionInstantiator ( ) [inline], [virtual]
```

0.15.239.2 Member Function Documentation

0.15.239.2.1 convert()

```
template<typename T >
bool smtrat::parser::IndexedFunctionInstantiator::convert (
            const std::vector< types::TermType > & from,
            std::vector< T > & to ) const [inline]
```

0.15.239.2.2 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]
```

0.15.240 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.

0.15.240.1 Detailed Description

Represents the i-th root of a multivariate polynomial at its main variable (given an appropriate sample).

0.15.240.2 Constructor & Destructor Documentation

0.15.240.2.1 IndexedRoot() [1/2]

```
smtrat::cadcells::datastructures::IndexedRoot::IndexedRoot (
    PolyRef p,
    size_t i ) [inline]
```

0.15.240.2.2 IndexedRoot() [2/2]

```
smtrat::cadcells::datastructures::IndexedRoot::IndexedRoot ( ) [inline]
```

0.15.240.3 Field Documentation

0.15.240.3.1 index

```
size_t smtrat::cadcells::datastructures::IndexedRoot::index
```

The index, must be > 0.

0.15.240.3.2 poly

```
PolyRef smtrat::cadcells::datastructures::IndexedRoot::poly
```

A multivariate polynomial.

0.15.241 smtrat::cadcells::datastructures::IndexedRootOrdering Class Reference

Describes an ordering of IndexedRoots.

```
#include <roots.h>
```

Public Member Functions

- void add_leq (RootFunction first, RootFunction second)
- void add_less (RootFunction first, RootFunction second)
- void add_eq (RootFunction first, RootFunction second)
- const auto & data () const
- const auto & leq () const
- const auto & geq () const
- bool holds_transitive (RootFunction first, RootFunction second, bool strict) const
- std::optional< RootFunction > holds_transitive (RootFunction first, PolyRef poly, bool strict) const
- std::optional< RootFunction > holds_transitive (PolyRef poly, RootFunction second, bool strict) const
- void polys (boost::container::flat_set< PolyRef > &result) const
- boost::container::flat_set< PolyRef > polys () const

0.15.241.1 Detailed Description

Describes an ordering of IndexedRoots.

0.15.241.2 Member Function Documentation

0.15.241.2.1 add_eq()

```
void smtrat::cadcells::datastructures::IndexedRootOrdering::add_eq (
    RootFunction first,
    RootFunction second ) [inline]
```

0.15.241.2.2 add_leq()

```
void smtrat::cadcells::datastructures::IndexedRootOrdering::add_leq (
    RootFunction first,
    RootFunction second ) [inline]
```

0.15.241.2.3 add_less()

```
void smtrat::cadcells::datastructures::IndexedRootOrdering::add_less (
    RootFunction first,
    RootFunction second ) [inline]
```

0.15.241.2.4 data()

```
const auto& smtrat::cadcells::datastructures::IndexedRootOrdering::data ( )
const [inline]
```

0.15.241.2.5 geq()

```
const auto& smtrat::cadcells::datastructures::IndexedRootOrdering::geq ( )
const [inline]
```

0.15.241.2.6 holds_transitive() [1/3]

```
std::optional<RootFunction> smtrat::cadcells::datastructures↩
::IndexedRootOrdering::holds_transitive (
    PolyRef poly,
    RootFunction second,
    bool strict ) const [inline]
```

0.15.241.2.7 holds_transitive() [2/3]

```
std::optional<RootFunction> smtrat::cadcells::datastructures↩
::IndexedRootOrdering::holds_transitive (
    RootFunction first,
    PolyRef poly,
    bool strict ) const [inline]
```

0.15.241.2.8 holds_transitive() [3/3]

```
bool smtrat::cadcells::datastructures::IndexedRootOrdering↩
::holds_transitive (
    RootFunction first,
    RootFunction second,
    bool strict ) const [inline]
```

0.15.241.2.9 leq()

```
const auto& smtrat::cadcells::datastructures::IndexedRootOrdering::leq ( )
const [inline]
```

0.15.241.2.10 polys() [1/2]

```
boost::container::flat_set<PolyRef> smtrat::cadcells::datastructures↩
::IndexedRootOrdering::polys ( ) const [inline]
```

0.15.241.2.11 polys() [2/2]

```
void smtrat::cadcells::datastructures::IndexedRootOrdering::polys (
    boost::container::flat_set< PolyRef > & result ) const [inline]
```

0.15.242 smtrat::cadcells::datastructures::IndexedRootRelation Struct Reference

A relation between two roots.

`#include <roots.h>`

Data Fields

- RootFunction first
- RootFunction second
- bool is_strict

0.15.242.1 Detailed Description

A relation between two roots.

0.15.242.2 Field Documentation

0.15.242.2.1 first `RootFunction smtrat::cadcells::datastructures::IndexedRootRelation::first`

0.15.242.2.2 is_strict `bool smtrat::cadcells::datastructures::IndexedRootRelation::is_strict`

0.15.242.2.3 second `RootFunction smtrat::cadcells::datastructures::IndexedRootRelation::second`

0.15.243 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

0.15.243.1 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

0.15.243.2 Member Typedef Documentation

0.15.243.2.1 CellEntry `template<class Settings >`

```
typedef std::pair<unsigned, Polynomial> smtrat::InequalitiesTable< Settings >::CellEntry
```

0.15.243.2.2 Ideal `template<class Settings >`

```
typedef Settings::MultivariateIdeal smtrat::InequalitiesTable< Settings >::Ideal  [protected]
```

0.15.243.2.3 Polynomial `template<class Settings >`

```
typedef Settings::PolynomialWithReasons smtrat::InequalitiesTable< Settings >::Polynomial
[protected]
```

0.15.243.2.4 RewriteRules `template<class Settings >`

```
typedef std::map<carl::Variable, std::pair<TermT, carl::BitVector> > smtrat::InequalitiesTable<
Settings >::RewriteRules
```

0.15.243.2.5 Row `template<class Settings >`

```
typedef std::pair<ModuleInput::const_iterator, RowEntry> smtrat::InequalitiesTable< Settings
>::Row
```

0.15.243.2.6 RowEntry `template<class Settings >`

```
typedef std::tuple<ModuleInput::iterator, carl::Relation, std::list<CellEntry> > smtrat::InequalitiesTable<
Settings >::RowEntry
```

0.15.243.2.7 Rows `template<class Settings >`

```
typedef std::map<ModuleInput::const_iterator, RowEntry, ModuleInput::IteratorCompare> smtrat::InequalitiesTab
Settings >::Rows
```

0.15.243.3 Constructor & Destructor Documentation

0.15.243.3.1 InequalitiesTable()

```
template<class Settings >
smtrat::InequalitiesTable< Settings >::InequalitiesTable (
            GBModule< Settings > * module )
```

0.15.243.4 Member Function Documentation

0.15.243.4.1 InsertReceivedFormula()

```
template<class Settings >
Rows::iterator smtrat::InequalitiesTable< Settings >::InsertReceivedFormula (
            ModuleInput::const_iterator received )
```

0.15.243.4.2 popBacktrackPoint()

```
template<class Settings >
void smtrat::InequalitiesTable< Settings >::popBacktrackPoint (
            unsigned nrBacktracks )
```

0.15.243.4.3 print()

```
template<class Settings >
void smtrat::InequalitiesTable< Settings >::print (
            std::ostream & os = std::cout ) const
```

0.15.243.4.4 pushBacktrackPoint()

```
template<class Settings >
void smtrat::InequalitiesTable< Settings >::pushBacktrackPoint ( )
```

0.15.243.4.5 reduceWRTGroebnerBasis() [1/3]

```
template<class Settings >
Answer smtrat::InequalitiesTable< Settings >::reduceWRTGroebnerBasis (
            const Ideal & gb,
            const RewriteRules & rules )
```

0.15.243.4.6 reduceWRTGroebnerBasis() [2/3]

```
template<class Settings >
Answer smtrat::InequalitiesTable< Settings >::reduceWRTGroebnerBasis (
            const std::list< typename Rows::iterator > & ineqToBeReduced,
            const Ideal & gb,
            const RewriteRules & rules )
```

0.15.243.4.7 reduceWRTGroebnerBasis() [3/3]

```
template<class Settings >
bool smtrat::InequalitiesTable< Settings >::reduceWRTGroebnerBasis (
            typename Rows::iterator ,
            const Ideal & gb,
            const RewriteRules & rules )
```

0.15.243.4.8 reduceWRTVariableRewriteRules() [1/3]

```
template<class Settings >
Answer smtrat::InequalitiesTable< Settings >::reduceWRTVariableRewriteRules (
            const RewriteRules & rules )
```

0.15.243.4.9 reduceWRTVariableRewriteRules() [2/3] template<class Settings >

```
Answer smtrat::InequalitiesTable< Settings >::reduceWRTVariableRewriteRules (
            const std::list< typename Rows::iterator > & ineqToBeReduced,
            const RewriteRules & rules )
```

0.15.243.4.10 reduceWRTVariableRewriteRules() [3/3] template<class Settings >

```
bool smtrat::InequalitiesTable< Settings >::reduceWRTVariableRewriteRules (
            typename Rows::iterator it,
            const RewriteRules & rules )
```

0.15.243.4.11 removeInequality() template<class Settings >

```
void smtrat::InequalitiesTable< Settings >::removeInequality (
            ModuleInput::const_iterator _formula )
```

0.15.243.5 Field Documentation

0.15.243.5.1 mBtnumber template<class Settings >

```
unsigned smtrat::InequalitiesTable< Settings >::mBtnumber [protected]
```

The actual number of backtrackpoints.

0.15.243.5.2 mLastRestart template<class Settings >

```
unsigned smtrat::InequalitiesTable< Settings >::mLastRestart [protected]
```

0.15.243.5.3 mModule template<class Settings >

```
GBModule<Settings>* smtrat::InequalitiesTable< Settings >::mModule [protected]
```

A pointer to the GBModule which uses this table.

0.15.243.5.4 mNewConstraints template<class Settings >

```
Rows::iterator smtrat::InequalitiesTable< Settings >::mNewConstraints [protected]
```

0.15.243.5.5 mReducedInequalities template<class Settings >

```
Rows smtrat::InequalitiesTable< Settings >::mReducedInequalities [protected]
```

A map of pointers from received iterators to rows.

0.15.244 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

0.15.244.1 Detailed Description

template<typename T1, typename T2>
struct smtrat::lra::Bound< T1, T2 >::Info

Stores some additional information for a bound.

0.15.244.2 Constructor & Destructor Documentation

0.15.244.2.1 Info()

```
template<typename T1 , typename T2 >
smtrat::lra::Bound< T1, T2 >::Info::Info (
            ModuleInput::iterator _position )  [inline]
```

0.15.244.3 Field Documentation

0.15.244.3.1 complement

```
template<typename T1 , typename T2 >
const Bound<T1, T2>* smtrat::lra::Bound< T1, T2 >::Info::complement
```

0.15.244.3.2 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.

0.15.244.3.3 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.

0.15.244.3.4 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.

0.15.244.3.5 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).

0.15.245 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

0.15.245.1 Field Documentation

0.15.245.1.1 abstractVariable `std::function<Minisat::Var(const FormulaT&)> smtrat::mcsat::InformationGetter::abstractVariable`

0.15.245.1.2 getBoolVarValue `std::function<Minisat::lbool(Minisat::Var)> smtrat::mcsat::InformationGetter::getBoolVarValue`

0.15.245.1.3 getClause `std::function<const Minisat::Clause&(Minisat::CRef)> smtrat::mcsat::InformationGetter::getClause`

0.15.245.1.4 getClauses `std::function<const Minisat::vec<Minisat::CRef>&()> smtrat::mcsat::InformationGetter::getClauses`

0.15.245.1.5 getDecisionLevel `std::function<int(Minisat::Var)> smtrat::mcsat::InformationGetter::getDecisionLevel`

0.15.245.1.6 getLearntClauses `std::function<const Minisat::vec<Minisat::CRef>&()> smtrat::mcsat::InformationGetter::getLearntClauses`

0.15.245.1.7 getLitValue `std::function<Minisat::lbool(Minisat::Lit)> smtrat::mcsat::InformationGetter::getLitValue`

0.15.245.1.8 getReason `std::function<Minisat::CRef(Minisat::Var)> smtrat::mcsat::InformationGetter::getReason`

0.15.245.1.9 getTrailIndex

`std::function<int(Minisat::Var)> smtrat::mcsat::InformationGetter::getTrailIndex`

0.15.245.1.10 getVarValue

`std::function<Minisat::lbool(Minisat::Var)> smtrat::mcsat::InformationGetter::getVarValue`

0.15.245.1.11 getWatches

`std::function<const Minisat::vec<Minisat::Watcher>&(Minisat::Lit)> smtrat::mcsat::InformationGetter::getWatches`

0.15.245.1.12 isAbstractedFormula

`std::function<bool(const FormulaT&)> smtrat::mcsat::InformationGetter::isAbstractedFormula`

0.15.245.1.13 isTheoryAbstraction

`std::function<bool(Minisat::Var)> smtrat::mcsat::InformationGetter::isTheoryAbstraction`

0.15.245.1.14 newVar

`std::function<Minisat::Var()> smtrat::mcsat::InformationGetter::newVar`

0.15.245.1.15 reabstractLiteral

`std::function<const FormulaT&(Minisat::Lit)> smtrat::mcsat::InformationGetter::reabstractLiteral`

0.15.245.1.16 reabstractVariable

`std::function<const FormulaT&(Minisat::Var)> smtrat::mcsat::InformationGetter::reabstractVariable`

0.15.246 smtrat::parser::Instantiator< V, T > Struct Template Reference

`#include <FunctionInstantiator.h>`

Public Member Functions

- template<typename Res >
  bool operator() (const Res &)
- bool operator() (carl::Variable v)
- template<typename TYPE = T>
  std::enable_if< std::is_same< TYPE, Poly >::value, bool >::type operator() (const Poly &p)
- bool operator() (const FormulaT &f)
- template<typename VAR = V>
  std::enable_if< std::is_same< VAR, types::BVVariable >::value, bool >::type operator() (const types::BVVariable &v)
- template<typename VAR = V>
  std::enable_if< std::is_same< VAR, types::BVVariable >::value, bool >::type operator() (const types::BVTerm &t)
- bool instantiate (V v, const T &repl, types::TermType &subject)

Protected Attributes

- V var
- T replacement
- types::TermType result

0.15.246.1 Member Function Documentation

0.15.246.1.1 instantiate()

```
template<typename V , typename T >
bool smtrat::parser::Instantiator< V, T >::instantiate (
            V v,
            const T & repl,
            types::TermType & subject )  [inline]
```

0.15.246.1.2 operator()() [1/6]

```
template<typename V , typename T >
bool smtrat::parser::Instantiator< V, T >::operator() (
            carl::Variable v )  [inline]
```

0.15.246.1.3 operator()() [2/6]

```
template<typename V , typename T >
bool smtrat::parser::Instantiator< V, T >::operator() (
            const FormulaT & f )  [inline]
```

0.15.246.1.4 operator()() [3/6]

```
template<typename V , typename T >
template<typename TYPE = T>
std::enable_if<std::is_same<TYPE,Poly>::value, bool>::type smtrat::parser::Instantiator< V,
T >::operator() (
            const Poly & p )  [inline]
```

0.15.246.1.5 operator()() [4/6]

```
template<typename V , typename T >
template<typename Res >
bool smtrat::parser::Instantiator< V, T >::operator() (
            const Res &  )  [inline]
```

0.15.246.1.6 operator()() [5/6]

```
template<typename V , typename T >
template<typename VAR = V>
std::enable_if<std::is_same<VAR,types::BVVariable>::value, bool>::type smtrat::parser::Instantiator<
V, T >::operator() (
            const types::BVTerm & t )  [inline]
```

0.15.246.1.7 operator()() [6/6]

```
template<typename V , typename T >
template<typename VAR = V>
std::enable_if<std::is_same<VAR,types::BVVariable>::value, bool>::type smtrat::parser::Instantiator<
V, T >::operator() (
            const types::BVVariable & v )  [inline]
```

0.15.246.2 Field Documentation

0.15.246.2.1 replacement

```
template<typename V , typename T >
T smtrat::parser::Instantiator< V, T >::replacement  [protected]
```

0.15.246.2.2 result `template<typename V , typename T >`

`types::TermType smtrat::parser::Instantiator< V, T >::result [protected]`

0.15.246.2.3 var `template<typename V , typename T >`

`V smtrat::parser::Instantiator< V, T >::var [protected]`

0.15.247 smtrat::parser::InstructionHandler Class Reference

`#include <InstructionHandler.h>`

Public Types

- typedef types::AttributeValue Value

Public Member Functions

- void addInstruction (std::function< void()> bind)
- bool hasInstructions () const
- void runInstructions ()
- void setArtificialVariables (std::vector< smtrat::ModelVariable > &&vars)
- void cleanModel (smtrat::Model &model) const
- bool has_info (const std::string &key) const
- const auto & get_info (const std::string &key) const
- template<typename T >
  T option (const std::string &key) const
- bool printInstruction () const
- InstructionHandler ()
- virtual ~InstructionHandler ()
- std::ostream & diagnostic ()
- void diagnostic (const std::string &s)
- std::ostream & regular ()
- void regular (const std::string &s)
- OutputWrapper error ()
- OutputWrapper warn ()
- OutputWrapper info ()
- virtual void add (const FormulaT &f)=0
- virtual void addSoft (const FormulaT &f, Rational weight, const std::string &id)=0
- virtual void annotateName (const FormulaT &f, const std::string &name)=0
- virtual void check ()=0
- virtual void declareFun (const carl::Variable &)=0
- virtual void declareSort (const std::string &, const unsigned &)=0
- virtual void defineSort (const std::string &, const std::vector< std::string > &, const carl::Sort &)=0
- virtual void echo (const std::string &s)
- virtual void eliminateQuantifiers (const qe::QEQuery &q)=0
- virtual void exit ()=0
- virtual void getAllModels ()=0
- virtual void getAssertions ()=0
- virtual void getAssignment ()=0
- void getInfo (const std::string &key)
- virtual void getModel ()=0
- virtual void getObjectives ()=0
- void getOption (const std::string &key)
- virtual void getProof ()=0
- virtual void getUnsatCore ()=0
- virtual void getValue (const std::vector< carl::Variable > &)=0

- virtual void addObjective (const Poly &p, OptimizationType ot)=0
- virtual void pop (std::size_t)=0
- virtual void push (std::size_t)=0
- virtual void reset ()
- virtual void resetAssertions ()=0
- void setInfo (const Attribute &attr)
- virtual void setLogic (const carl::Logic &)=0
- void setOption (const Attribute &option)

Protected Member Functions

- void setStream (const std::string &s, std::ostream &os)

Protected Attributes

- VariantMap< std::string, Value > infos
- VariantMap< std::string, Value > options
- std::ostream mRegular
- std::ostream mDiagnostic
- std::map< std::string, std::ofstream > streams

0.15.247.1 Member Typedef Documentation

0.15.247.1.1 Value `typedef types::AttributeValue smtrat::parser::InstructionHandler::Value`

0.15.247.2 Constructor & Destructor Documentation

0.15.247.2.1 InstructionHandler() `smtrat::parser::InstructionHandler::InstructionHandler ( ) [inline]`

0.15.247.2.2 ~InstructionHandler() `virtual smtrat::parser::InstructionHandler::~InstructionHandler ( ) [inline], [virtual]`

0.15.247.3 Member Function Documentation

0.15.247.3.1 add() `virtual void smtrat::parser::InstructionHandler::add ( const FormulaT & f ) [pure virtual]`

Implemented in smtrat::parseformula::FormulaCollector, and smtrat::Executor< Strategy >.

0.15.247.3.2 addInstruction() `void smtrat::parser::InstructionHandler::addInstruction ( std::function< void()> bind ) [inline]`

0.15.247.3.3 addObjective() `virtual void smtrat::parser::InstructionHandler::addObjective ( const Poly & p, OptimizationType ot ) [pure virtual]`

Implemented in smtrat::Executor< Strategy >, and smtrat::parseformula::FormulaCollector.

0.15.247.3.4 addSoft()

```
virtual void smtrat::parser::InstructionHandler::addSoft (
    const FormulaT & f,
    Rational weight,
    const std::string & id ) [pure virtual]
```

Implemented in smtrat::Executor< Strategy >, and smtrat::parseformula::FormulaCollector.

0.15.247.3.5 annotateName()

```
virtual void smtrat::parser::InstructionHandler::annotateName (
    const FormulaT & f,
    const std::string & name ) [pure virtual]
```

Implemented in smtrat::Executor< Strategy >, and smtrat::parseformula::FormulaCollector.

0.15.247.3.6 check()

```
virtual void smtrat::parser::InstructionHandler::check ( ) [pure virtual]
```

Implemented in smtrat::parseformula::FormulaCollector, and smtrat::Executor< Strategy >.

0.15.247.3.7 cleanModel()

```
void smtrat::parser::InstructionHandler::cleanModel (
    smtrat::Model & model ) const [inline]
```

0.15.247.3.8 declareFun()

```
virtual void smtrat::parser::InstructionHandler::declareFun (
    const carl::Variable & ) [pure virtual]
```

Implemented in smtrat::parseformula::FormulaCollector, and smtrat::Executor< Strategy >.

0.15.247.3.9 declareSort()

```
virtual void smtrat::parser::InstructionHandler::declareSort (
    const std::string & ,
    const unsigned & ) [pure virtual]
```

Implemented in smtrat::parseformula::FormulaCollector, and smtrat::Executor< Strategy >.

0.15.247.3.10 defineSort()

```
virtual void smtrat::parser::InstructionHandler::defineSort (
    const std::string & ,
    const std::vector< std::string > & ,
    const carl::Sort & ) [pure virtual]
```

Implemented in smtrat::parseformula::FormulaCollector, and smtrat::Executor< Strategy >.

0.15.247.3.11 diagnostic() [1/2]

```
std::ostream& smtrat::parser::InstructionHandler::diagnostic (
) [inline]
```

0.15.247.3.12 diagnostic() [2/2]

```
void smtrat::parser::InstructionHandler::diagnostic (
    const std::string & s ) [inline]
```

0.15.247.3.13 echo()

```
virtual void smtrat::parser::InstructionHandler::echo (
    const std::string & s ) [inline], [virtual]
```

0.15.247.3.14 eliminateQuantifiers()

```
virtual void smtrat::parser::InstructionHandler::eliminate↩
Quantifiers (
    const qe::QEQuery & q ) [pure virtual]
```

Implemented in smtrat::parseformula::FormulaCollector, and smtrat::Executor< Strategy >.

0.15.247.3.15 error() `OutputWrapper smtrat::parser::InstructionHandler::error ( ) [inline]`

0.15.247.3.16 exit() `virtual void smtrat::parser::InstructionHandler::exit ( ) [pure virtual]`

Implemented in smtrat::parseformula::FormulaCollector, and smtrat::Executor< Strategy >.

0.15.247.3.17 get_info() `const auto& smtrat::parser::InstructionHandler::get_info ( const std::string & key ) const [inline]`

0.15.247.3.18 getAllModels() `virtual void smtrat::parser::InstructionHandler::getAllModels ( ) [pure virtual]`

Implemented in smtrat::parseformula::FormulaCollector, and smtrat::Executor< Strategy >.

0.15.247.3.19 getAssertions() `virtual void smtrat::parser::InstructionHandler::getAssertions ( ) [pure virtual]`

Implemented in smtrat::parseformula::FormulaCollector, and smtrat::Executor< Strategy >.

0.15.247.3.20 getAssignment() `virtual void smtrat::parser::InstructionHandler::getAssignment ( ) [pure virtual]`

Implemented in smtrat::parseformula::FormulaCollector, and smtrat::Executor< Strategy >.

0.15.247.3.21 getInfo() `void smtrat::parser::InstructionHandler::getInfo ( const std::string & key ) [inline]`

0.15.247.3.22 getModel() `virtual void smtrat::parser::InstructionHandler::getModel ( ) [pure virtual]`

Implemented in smtrat::parseformula::FormulaCollector, and smtrat::Executor< Strategy >.

0.15.247.3.23 getObjectives() `virtual void smtrat::parser::InstructionHandler::getObjectives ( ) [pure virtual]`

Implemented in smtrat::parseformula::FormulaCollector, and smtrat::Executor< Strategy >.

0.15.247.3.24 getOption() `void smtrat::parser::InstructionHandler::getOption ( const std::string & key ) [inline]`

0.15.247.3.25 getProof() `virtual void smtrat::parser::InstructionHandler::getProof ( ) [pure virtual]`

Implemented in smtrat::parseformula::FormulaCollector, and smtrat::Executor< Strategy >.

0.15.247.3.26 getUnsatCore() `virtual void smtrat::parser::InstructionHandler::getUnsatCore ( ) [pure virtual]`

Implemented in smtrat::parseformula::FormulaCollector, and smtrat::Executor< Strategy >.

0.15.247.3.27 getValue()

`virtual void smtrat::parser::InstructionHandler::getValue ( const std::vector< carl::Variable > & ) [pure virtual]`

Implemented in smtrat::parseformula::FormulaCollector, and smtrat::Executor< Strategy >.

0.15.247.3.28 has_info()

`bool smtrat::parser::InstructionHandler::has_info ( const std::string & key ) const [inline]`

0.15.247.3.29 hasInstructions()

`bool smtrat::parser::InstructionHandler::hasInstructions ( ) const [inline]`

0.15.247.3.30 info()

`OutputWrapper smtrat::parser::InstructionHandler::info ( ) [inline]`

0.15.247.3.31 option()

`template<typename T > T smtrat::parser::InstructionHandler::option ( const std::string & key ) const [inline]`

0.15.247.3.32 pop()

`virtual void smtrat::parser::InstructionHandler::pop ( std::size_t ) [pure virtual]`

Implemented in smtrat::parseformula::FormulaCollector, and smtrat::Executor< Strategy >.

0.15.247.3.33 printInstruction()

`bool smtrat::parser::InstructionHandler::printInstruction ( ) const [inline]`

0.15.247.3.34 push()

`virtual void smtrat::parser::InstructionHandler::push ( std::size_t ) [pure virtual]`

Implemented in smtrat::parseformula::FormulaCollector, and smtrat::Executor< Strategy >.

0.15.247.3.35 regular() [1/2]

`std::ostream& smtrat::parser::InstructionHandler::regular ( ) [inline]`

0.15.247.3.36 regular() [2/2]

`void smtrat::parser::InstructionHandler::regular ( const std::string & s ) [inline]`

0.15.247.3.37 reset()

`virtual void smtrat::parser::InstructionHandler::reset ( ) [inline], [virtual]`

Reimplemented in smtrat::parseformula::FormulaCollector, and smtrat::Executor< Strategy >.

0.15.247.3.38 resetAssertions()

`virtual void smtrat::parser::InstructionHandler::resetAssertions ( ) [pure virtual]`

Implemented in smtrat::parseformula::FormulaCollector, and smtrat::Executor< Strategy >.

0.15.247.3.39 runInstructions()

`void smtrat::parser::InstructionHandler::runInstructions ( ) [inline]`

0.15.247.3.40 setArtificialVariables()

```
void smtrat::parser::InstructionHandler::setArtificial↩
Variables (
            std::vector< smtrat::ModelVariable > && vars ) [inline]
```

0.15.247.3.41 setInfo()

```
void smtrat::parser::InstructionHandler::setInfo (
            const Attribute & attr ) [inline]
```

0.15.247.3.42 setLogic()

```
virtual void smtrat::parser::InstructionHandler::setLogic (
            const carl::Logic & ) [pure virtual]
```

Implemented in smtrat::Executor< Strategy >, and smtrat::parseformula::FormulaCollector.

0.15.247.3.43 setOption()

```
void smtrat::parser::InstructionHandler::setOption (
            const Attribute & option ) [inline]
```

0.15.247.3.44 setStream()

```
void smtrat::parser::InstructionHandler::setStream (
            const std::string & s,
            std::ostream & os ) [inline], [protected]
```

0.15.247.3.45 warn()

```
OutputWrapper smtrat::parser::InstructionHandler::warn ( ) [inline]
```

0.15.247.4 Field Documentation

0.15.247.4.1 infos

```
VariantMap<std::string, Value> smtrat::parser::InstructionHandler::infos
[protected]
```

0.15.247.4.2 mDiagnostic

```
std::ostream smtrat::parser::InstructionHandler::mDiagnostic [protected]
```

0.15.247.4.3 mRegular

```
std::ostream smtrat::parser::InstructionHandler::mRegular [protected]
```

0.15.247.4.4 options

```
VariantMap<std::string, Value> smtrat::parser::InstructionHandler::options
[protected]
```

0.15.247.4.5 streams

```
std::map<std::string, std::ofstream> smtrat::parser::InstructionHandler↩
::streams [protected]
```

0.15.248 Minisat::Int64Range Struct Reference

```
#include <Options.h>
```

Public Member Functions

- Int64Range (int64_t b, int64_t e)

Data Fields

- int64_t begin
- int64_t end

0.15.248.1 Constructor & Destructor Documentation

0.15.248.1.1 Int64Range()

```
Minisat::Int64Range::Int64Range (
int64_t b,
int64_t e ) [inline]
```

0.15.248.2 Field Documentation

0.15.248.2.1 begin

```
int64_t Minisat::Int64Range::begin
```

0.15.248.2.2 end

```
int64_t Minisat::Int64Range::end
```

0.15.249 smtrat::IntBlastModule< Settings > Class Template Reference

```
#include <IntBlastModule.h>
```

Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- IntBlastModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=NULL)
- ~IntBlastModule ()
- bool informCore (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void init ()

  Informs all backends about the so far encountered constraints, which have not yet been communicated.
- bool addCore (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- void removeCore (ModuleInput::const_iterator _subformula)

  Removes the subformula of the received formula at the given position to the considered ones of this module.
- void updateModel () const

  Updates the current assignment into the model.
- Answer checkCore ()

  Checks the received formula for consistency.
- bool inform (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void deinform (const FormulaT &_constraint)

  The inverse of informing about a constraint.
- bool add (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- virtual Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_↩ VARIABLE)

Checks the received formula for consistency.

- virtual void remove (ModuleInput::const_iterator _subformula)

  Removes everything related to the given sub-formula of the received formula.
- virtual void updateAllModels ()

  Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
- virtual std::list< std::vector< carl::Variable > > getModelEqualities () const

  Partition the variables from the current model into equivalence classes according to their assigned value.
- unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const
- virtual unsigned currentlySatisfied (const FormulaT &) const
- bool receivedVariable (carl::Variable::Arg _var) const
- Answer solverState () const
- std::size_t id () const
- void setId (std::size_t _id)

  Sets this modules unique ID to identify itself.
- thread_priority threadPriority () const
- void setThreadPriority (thread_priority _threadPriority)

  Sets the priority of this module to get a thread for running its check procedure.
- const ModuleInput * pReceivedFormula () const
- const ModuleInput & rReceivedFormula () const
- const ModuleInput * pPassedFormula () const
- const ModuleInput & rPassedFormula () const
- const Model & model () const
- const std::vector< Model > & allModels () const
- const std::vector< FormulaSetT > & infeasibleSubsets () const
- const std::vector< Module * > & usedBackends () const
- const carl::FastSet< FormulaT > & constraintsToInform () const
- const carl::FastSet< FormulaT > & informedConstraints () const
- void addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))

  Stores a lemma being a valid formula.
- bool hasLemmas ()

  Checks whether this module or any of its backends provides any lemmas.
- void clearLemmas ()

  Deletes all yet found lemmas.
- const std::vector< Lemma > & lemmas () const
- ModuleInput::const_iterator firstUncheckedReceivedSubformula () const
- ModuleInput::const_iterator firstSubformulaToPass () const
- void receivedFormulaChecked ()

  Notifies that the received formulas has been checked.
- const 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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.
- void cleanModel () const

Substitutes variable occurrences by its model value in the model values of other variables.

- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)
- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

  Copies the infeasible subsets of the passed formula.
- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.

- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.249.1 Member Typedef Documentation

0.15.249.1.1 SettingsType

```
template<class Settings >
typedef Settings smtrat::IntBlastModule< Settings >::SettingsType
```

0.15.249.2 Member Enumeration Documentation

0.15.249.2.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

0.15.249.3 Constructor & Destructor Documentation

0.15.249.3.1 IntBlastModule()

```
template<class Settings >
smtrat::IntBlastModule< Settings >::IntBlastModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = NULL )
```

0.15.249.3.2 ~IntBlastModule() template<class Settings >

smtrat::IntBlastModule< Settings >::~IntBlastModule ( )

0.15.249.4 Member Function Documentation

0.15.249.4.1 add() bool smtrat::Module::add (ModuleInput::const_iterator _subformula) [inherited]

The module has to take the given sub-formula of the received formula into account.

Parameters

_subformula	The sub-formula to take additionally into account.

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

0.15.249.4.2 addConstraintToInform() void smtrat::Module::addConstraintToInform (const FormulaT & _constraint) [protected], [virtual], [inherited]

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

0.15.249.4.3 addCore() [1/2] virtual bool smtrat::Module::addCore ([[maybe_unused]] 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.

0.15.249.4.4 addCore() [2/2] template<class Settings >

bool smtrat::IntBlastModule< Settings >::addCore (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.

0.15.249.4.5 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.249.4.6 addLemma()

```
void smtrat::Module::addLemma (
    const FormulaT & _lemma,
    const LemmaType & _lt = LemmaType::NORMAL,
    const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline],
[inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.249.4.7 addOrigin()

```
void smtrat::Module::addOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.249.4.8 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.249.4.9 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula )  [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.249.4.10 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula,
            const FormulaT & _origin )  [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.249.4.11 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.

0.15.249.4.12 allModels() `const std::vector<Model>& smtrat::Module::allModels ( ) const [inline], [inherited]`

Returns

All satisfying assignments, if existent.

0.15.249.4.13 anAnswerFound() `bool smtrat::Module::anAnswerFound ( ) const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.249.4.14 answerFound() `const smtrat::Conditionals& smtrat::Module::answerFound ( ) const [inline], [inherited]`

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.249.4.15 backendsModel() `const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.249.4.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.249.4.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.249.4.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
```

```
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline],
[protected], [inherited]
```

0.15.249.4.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
const Poly & _polynomial,
bool _integral,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.249.4.20 check()

```
Answer smtrat::Module::check (
bool _final = false,
bool _full = true,
carl::Variable _objective = carl::Variable::NO_VARIABLE ) [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.249.4.21 checkCore() `template<class Settings >`

`Answer smtrat::IntBlastModule< Settings >::checkCore ( ) [virtual]`

Checks the received formula for consistency.

Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.249.4.22 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.

0.15.249.4.23 checkModel()

`unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]`

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.249.4.24 cleanModel()

`void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.249.4.25 clearLemmas()

`void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

0.15.249.4.26 clearModel()

`void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

0.15.249.4.27 clearModels()

`void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

0.15.249.4.28 clearPassedFormula()

`void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.249.4.29 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

0.15.249.4.30 collectOrigins() [2/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.
If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.249.4.31 collectTheoryPropagations()

```
void smtrat::Module::collectTheoryPropagations ( ) [inherited]
```

0.15.249.4.32 constraintsToInform()

```
const carl::FastSet<FormulaT>& smtrat::Module::constraints↩
ToInform ( ) const [inline], [inherited]
```

Returns

The constraints which the backends must be informed about.

0.15.249.4.33 currentlySatisfied()

```
virtual unsigned smtrat::Module::currentlySatisfied (
    const FormulaT & ) const [inline], [virtual], [inherited]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.249.4.34 currentlySatisfiedByBackend()

```
unsigned smtrat::Module::currentlySatisfiedByBackend (
    const FormulaT & _formula ) const [inherited]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.249.4.35 deinform()

```
void smtrat::Module::deinform (
    const FormulaT & _constraint ) [inherited]
```

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.249.4.36 deinformCore()

`virtual void smtrat::Module::deinformCore ( [[maybe_unused] ] 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.

0.15.249.4.37 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

0.15.249.4.38 eraseSubformulaFromPassedFormula()

`ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula ( ModuleInput::iterator _subformula, bool _ignoreOrigins = false ) [protected], [virtual], [inherited]`

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_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 >.

0.15.249.4.39 excludeNotReceivedVariablesFromModel()

`void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

0.15.249.4.40 findBestOrigin()

`std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin ( const std::vector< FormulaT > & _origins ) const [protected], [inherited]`

Parameters

_origins	

Returns

0.15.249.4.41 firstSubformulaToPass()

`ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.249.4.42 firstUncheckedReceivedSubformula()

`ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.249.4.43 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.249.4.44 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.249.4.45 getBackendsAllModels()

`void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.249.4.46 getBackendsModel()

`void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.249.4.47 getInfeasibleSubsets() [1/2]

`void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.249.4.48 getInfeasibleSubsets() [2/2]

```
std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets (
            const Module & _backend ) const  [protected], [inherited]
```

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

Parameters

_backend	The backend from which to obtain the infeasible subsets.

Returns

The infeasible subsets the given backend provides.

0.15.249.4.49 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.249.4.50 getModelEqualities()

`std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.249.4.51 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
const FormulaT & _formula,
FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.249.4.52 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
const FormulaT & _formula,
FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.249.4.53 getOrigins() [3/3]

```
const FormulaT& smtrat::Module::getOrigins (
ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]
```

Gets the origins of the passed formula at the given position.

Parameters

_formula	The position of a formula in the passed formulas.

Returns

The origins of the passed formula at the given position.

0.15.249.4.54 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.

0.15.249.4.55 hasLemmas() `bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

0.15.249.4.56 hasValidInfeasibleSubset() `bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]`

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.249.4.57 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.249.4.58 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.249.4.59 inform() `bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.
It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

Parameters

_constraint	The constraint to inform about.

Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

0.15.249.4.60 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.

0.15.249.4.61 informCore() [1/2]

`virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.249.4.62 informCore() [2/2]

`template<class Settings > bool smtrat::IntBlastModule< Settings >::informCore ( 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.

0.15.249.4.63 informedConstraints()

`const carl::FastSet<FormulaT>& smtrat::Module::informedConstraints ( ) const [inline], [inherited]`

Returns

The position of the first constraint of which no backend has been informed about.

0.15.249.4.64 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.

0.15.249.4.65 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.249.4.66 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.

0.15.249.4.67 isPreprocessor() `bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.249.4.68 lemmas() `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.249.4.69 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.

0.15.249.4.70 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.249.4.71 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.

0.15.249.4.72 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.

0.15.249.4.73 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

0.15.249.4.74 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.249.4.75 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.249.4.76 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.249.4.77 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.249.4.78 pReceivedFormula() `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.249.4.79 print() `void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.249.4.80 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.

0.15.249.4.81 printInfeasibleSubsets() `void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.249.4.82 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.

0.15.249.4.83 printPassedFormula() `void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.249.4.84 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.249.4.85 probablyLooping()

`bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.249.4.86 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.249.4.87 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

0.15.249.4.88 receivedVariable()

`bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

0.15.249.4.89 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.

0.15.249.4.90 removeCore() [1/2]

```
virtual void smtrat::Module::removeCore (
            [[maybe_unused] ] 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.

0.15.249.4.91 removeCore() [2/2]

```
template<class Settings >
void smtrat::IntBlastModule< Settings >::removeCore (
            ModuleInput::const_iterator _subformula )
```

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.

0.15.249.4.92 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origin (
            ModuleInput::iterator _formula,
            const FormulaT & _origin ) [inline], [protected], [inherited]
```

0.15.249.4.93 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

0.15.249.4.94 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.249.4.95 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to this module.

0.15.249.4.96 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( ) [inline],
[protected], [virtual], [inherited]
```

Reimplemented in smtrat::PModule.

0.15.249.4.97 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.

0.15.249.4.98 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.

0.15.249.4.99 setThreadPriority()

```
void smtrat::Module::setThreadPriority (
    thread_priority _threadPriority ) [inline], [inherited]
```

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.249.4.100 solverState()

```
Answer smtrat::Module::solverState ( ) const [inline], [inherited]
```

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.249.4.101 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 !=.

0.15.249.4.102 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.249.4.103 updateAllModels() `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.249.4.104 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.249.4.105 updateModel() `template<class Settings > void smtrat::IntBlastModule< Settings >::updateModel ( ) const [virtual]`

Updates the current assignment into the model.
Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.
Reimplemented from smtrat::Module.

0.15.249.4.106 usedBackends() `const std::vector<Module*>& smtrat::Module::usedBackends ( ) const [inline], [inherited]`

Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.249.5 Field Documentation

0.15.249.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.249.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.249.5.3 mBackendsFoundAnswer `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.249.5.4 mConstraintsToInform

`carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.249.5.5 mFinalCheck

`bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.249.5.6 mFirstPosInLastBranches

`std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.249.5.7 mFirstSubformulaToPass

`ModuleInput::iterator smtrat::Module::mFirstSubformulaTo↩ Pass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.249.5.8 mFirstUncheckedReceivedSubformula

`ModuleInput::const_iterator smtrat::Module↩ ::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.249.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.249.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.249.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.249.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.249.5.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std↩ ::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.249.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.249.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.249.5.16 mModelComputed `bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.249.5.17 mNumOfBranchVarsToStore `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.249.5.18 mObjectiveVariable `carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.249.5.19 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

0.15.249.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.249.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.249.5.22 mSolverState `std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.249.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

0.15.249.5.24 mUsedBackends `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.249.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.250 smtrat::IntEqModule< Settings > Class Template Reference

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

`#include <IntEqModule.h>`

Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- IntEqModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=NULL)
- ~IntEqModule ()
- bool addCore (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- void removeCore (ModuleInput::const_iterator _subformula)

  Removes the subformula of the received formula at the given position to the considered ones of this module.
- void updateModel () const

  Updates the current assignment into the model.
- Answer checkCore ()

  Checks the received formula for consistency.
- bool inform (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void deinform (const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual void init ()

  Informs all backends about the so far encountered constraints, which have not yet been communicated.
- bool add (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- virtual Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_↩VARIABLE)

  Checks the received formula for consistency.
- virtual void remove (ModuleInput::const_iterator _subformula)

  Removes everything related to the given sub-formula of the received formula.
- virtual void updateAllModels ()

  Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
- virtual std::list< std::vector< carl::Variable > > getModelEqualities () const

  Partition the variables from the current model into equivalence classes according to their assigned value.
- unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const
- virtual unsigned currentlySatisfied (const FormulaT &) const
- bool receivedVariable (carl::Variable::Arg _var) const
- Answer solverState () const
- std::size_t id () const
- void setId (std::size_t _id)

  Sets this modules unique ID to identify itself.
- thread_priority threadPriority () const
- void setThreadPriority (thread_priority _threadPriority)

  Sets the priority of this module to get a thread for running its check procedure.
- const ModuleInput * pReceivedFormula () const
- const ModuleInput & rReceivedFormula () const
- const ModuleInput * pPassedFormula () const
- const ModuleInput & rPassedFormula () const
- const Model & model () const
- const std::vector< Model > & allModels () const
- const std::vector< FormulaSetT > & infeasibleSubsets () const
- const std::vector< Module * > & usedBackends () const
- const carl::FastSet< FormulaT > & constraintsToInform () const
- const carl::FastSet< FormulaT > & informedConstraints () const
- void addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))

  Stores a lemma being a valid formula.

- bool hasLemmas ()

  Checks whether this module or any of its backends provides any lemmas.
- void clearLemmas ()

  Deletes all yet found lemmas.
- const std::vector< Lemma > & lemmas () const
- ModuleInput::const_iterator firstUncheckedReceivedSubformula () const
- ModuleInput::const_iterator firstSubformulaToPass () const
- void receivedFormulaChecked ()

  Notifies that the received formulas has been checked.
- const 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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.
- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)
- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.

- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

  Copies the infeasible subsets of the passed formula.
- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.250.1 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.

0.15.250.2 Member Typedef Documentation

0.15.250.2.1 SettingsType `template<typename Settings >`

`typedef Settings smtrat::IntEqModule< Settings >::SettingsType`

0.15.250.3 Member Enumeration Documentation

0.15.250.3.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

0.15.250.4 Constructor & Destructor Documentation

0.15.250.4.1 IntEqModule() `template<typename Settings >`

```
smtrat::IntEqModule< Settings >::IntEqModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = NULL )
```

0.15.250.4.2 ~IntEqModule() `template<typename Settings >`

`smtrat::IntEqModule< Settings >::~IntEqModule ( ) [inline]`

0.15.250.5 Member Function Documentation

0.15.250.5.1 add()

```
bool smtrat::Module::add (
            ModuleInput::const_iterator _subformula ) [inherited]
```

The module has to take the given sub-formula of the received formula into account.

Parameters

_subformula	The sub-formula to take additionally into account.

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

0.15.250.5.2 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 >.

0.15.250.5.3 addCore() [1/2]

```
virtual bool smtrat::Module::addCore (
    [[maybe_unused] ] 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.

0.15.250.5.4 addCore() [2/2]

```
template<typename Settings >
bool smtrat::IntEqModule< Settings >::addCore (
    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.

0.15.250.5.5 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula
(
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.250.5.6 addLemma()

```
void smtrat::Module::addLemma (
    const FormulaT & _lemma,
    const LemmaType & _lt = LemmaType::NORMAL,
    const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline],
[inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.250.5.7 addOrigin()

```
void smtrat::Module::addOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.250.5.8 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.250.5.9 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.250.5.10 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.250.5.11 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.

0.15.250.5.12 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns

All satisfying assignments, if existent.

0.15.250.5.13 anAnswerFound()

```
bool smtrat::Module::anAnswerFound ( ) const [inline], [protected],
[inherited]
```

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.250.5.14 answerFound()

```
const smtrat::Conditionals& smtrat::Module::answerFound ( ) const
[inline], [inherited]
```

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.250.5.15 backendsModel()

```
const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]
```

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.250.5.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.250.5.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.250.5.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.250.5.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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.

Parameters

_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

0.15.250.5.20 check()

```
Answer smtrat::Module::check (
    bool _final = false,
    bool _full = true,
    carl::Variable _objective = carl::Variable::NO_VARIABLE ) [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.250.5.21 checkCore()

```
template<typename Settings >
Answer smtrat::IntEqModule< Settings >::checkCore ( ) [virtual]
```

Checks the received formula for consistency.

Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.250.5.22 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.

0.15.250.5.23 checkModel() `unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]`

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.250.5.24 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.250.5.25 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

0.15.250.5.26 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

0.15.250.5.27 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

0.15.250.5.28 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.250.5.29 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

0.15.250.5.30 collectOrigins() [2/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.
If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.250.5.31 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

0.15.250.5.32 constraintsToInform() `const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns

The constraints which the backends must be informed about.

0.15.250.5.33 currentlySatisfied() `virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.250.5.34 currentlySatisfiedByBackend() `unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.250.5.35 deinform() `void smtrat::Module::deinform ( const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.250.5.36 deinformCore() `virtual void smtrat::Module::deinformCore ( [[maybe_unused] ] 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.

0.15.250.5.37 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

0.15.250.5.38 eraseSubformulaFromPassedFormula()

```
ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (
            ModuleInput::iterator _subformula,
            bool _ignoreOrigins = false ) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_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 >.

0.15.250.5.39 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

0.15.250.5.40 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
            const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

0.15.250.5.41 firstSubformulaToPass()

```
ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]
```

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.250.5.42 firstUncheckedReceivedSubformula() `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.250.5.43 freeSplittingVariable() `static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.250.5.44 generateTrivialInfeasibleSubset() `void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.250.5.45 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.250.5.46 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.250.5.47 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.250.5.48 getInfeasibleSubsets() [2/2] `std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets ( const Module & _backend ) const [protected], [inherited]`

Get the infeasible subsets the given backend provides.
Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

Parameters

_backend	The backend from which to obtain the infeasible subsets.

Returns

The infeasible subsets the given backend provides.

0.15.250.5.49 getInformationRelevantFormulas() `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.250.5.50 getModelEqualities() `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.250.5.51 getOrigins() [1/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.250.5.52 getOrigins() [2/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.250.5.53 getOrigins() [3/3] `const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]`

Gets the origins of the passed formula at the given position.

Parameters

_formula	The position of a formula in the passed formulas.

Returns

The origins of the passed formula at the given position.

0.15.250.5.54 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.

0.15.250.5.55 hasLemmas() `bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

0.15.250.5.56 hasValidInfeasibleSubset() `bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]`

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.250.5.57 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.250.5.58 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasible↩ Subsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.250.5.59 inform() `bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.
It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

Parameters

_constraint	The constraint to inform about.

Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

0.15.250.5.60 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.

0.15.250.5.61 informCore() `virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.250.5.62 informedConstraints() `const carl::FastSet<FormulaT>& smtrat::Module::informedConstraints ( ) const [inline], [inherited]`

Returns

The position of the first constraint of which no backend has been informed about.

0.15.250.5.63 init() `void smtrat::Module::init ( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not be called twice and only before the first runBackends call.
Reimplemented in smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntBlastModule< Settings >, smtrat::FPPModule< Settings >, smtrat::CurryModule< Settings >, and smtrat::BVModule< Settings >.

0.15.250.5.64 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.250.5.65 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.

0.15.250.5.66 isPreprocessor() `bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.250.5.67 lemmas() `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.250.5.68 merge()

```
std::vector< FormulaT > smtrat::Module::merge (
    const std::vector< FormulaT > & _vecSetA,
    const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]
```

Merges the two vectors of sets into the first one.
({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

Parameters

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns

The vector being the two given vectors merged.

0.15.250.5.69 model()

```
const Model& smtrat::Module::model ( ) const [inline], [inherited]
```

Returns

The assignment of the current satisfiable result, if existent.

0.15.250.5.70 modelsDisjoint()

```
bool smtrat::Module::modelsDisjoint (
    const Model & _modelA,
    const Model & _modelB ) [static], [protected], [inherited]
```

Checks whether there is no variable assigned by both models.

Parameters

_modelA	The first model to check for.
_modelB	The second model to check for.

Returns

true, if there is no variable assigned by both models; false, otherwise.

0.15.250.5.71 moduleName()

```
template<typename Settings >
std::string smtrat::IntEqModule< Settings >::moduleName ( ) const [inline], [virtual]
```

Returns

The name of the given module type as name.

Reimplemented from smtrat::Module.

0.15.250.5.72 objective()

```
carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]
```

0.15.250.5.73 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.250.5.74 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.250.5.75 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.250.5.76 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.250.5.77 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.250.5.78 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.250.5.79 printAllModels()

`void smtrat::Module::printAllModels ( std::ostream & _out = std::cout ) [inherited]`

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.250.5.80 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
const std::string & _initiation = "***" ) const [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.250.5.81 printModel()

```
void smtrat::Module::printModel (
std::ostream & _out = std::cout ) const [inherited]
```

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.250.5.82 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.250.5.83 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.250.5.84 probablyLooping()

```
bool smtrat::Module::probablyLooping (
const typename Poly::PolyType & _branchingPolynomial,
const Rational & _branchingValue ) const [protected], [inherited]
```

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.250.5.85 receivedFormulaChecked() `void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.250.5.86 receivedFormulasAsInfeasibleSubset() `void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

0.15.250.5.87 receivedVariable() `bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

0.15.250.5.88 remove() `void smtrat::Module::remove ( ModuleInput::const_iterator _subformula ) [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.
However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

Parameters

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

0.15.250.5.89 removeCore() [1/2] `virtual void smtrat::Module::removeCore ( [[maybe_unused] ] 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.

0.15.250.5.90 removeCore() [2/2] `template<typename Settings > void smtrat::IntEqModule< Settings >::removeCore ( ModuleInput::const_iterator _subformula )`

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.

0.15.250.5.91 removeOrigin() `std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩Origin (`

```
ModuleInput::iterator _formula,
const FormulaT & _origin )  [inline], [protected], [inherited]
```

0.15.250.5.92 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins (
ModuleInput::iterator _formula,
const std::shared_ptr< std::vector< FormulaT >> & _origins )  [inline], [protected], [inherited]
```

0.15.250.5.93 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.250.5.94 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]
```

Returns

A reference to the formula passed to this module.

0.15.250.5.95 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( )  [inline], [protected], [virtual], [inherited]
```

Reimplemented in smtrat::PModule.

0.15.250.5.96 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.

0.15.250.5.97 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.

0.15.250.5.98 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.

0.15.250.5.99 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.

0.15.250.5.100 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 !=.

0.15.250.5.101 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.250.5.102 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 >.

0.15.250.5.103 updateLemmas()

`void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.250.5.104 updateModel() `template<typename Settings > void smtrat::IntEqModule< Settings >::updateModel ( ) const [virtual]`

Updates the current assignment into the model.
Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.
Reimplemented from smtrat::Module.

0.15.250.5.105 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).

0.15.250.6 Field Documentation

0.15.250.6.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.250.6.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.250.6.3 mBackendsFoundAnswer `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.250.6.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.250.6.5 mFinalCheck `bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.250.6.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.250.6.7 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.

0.15.250.6.8 mFirstUncheckedReceivedSubformula `ModuleInput::const_iterator smtrat::Module↩ ::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.250.6.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.250.6.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.250.6.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.250.6.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.250.6.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std↩::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.250.6.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.250.6.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.250.6.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.250.6.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.250.6.18 mObjectiveVariable

`carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.250.6.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplitting↩Variables [static], [inherited]`

Reusable splitting variables.

0.15.250.6.20 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.250.6.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.250.6.22 mSolverState `std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.250.6.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

0.15.250.6.24 mUsedBackends `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.250.6.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.251 smtrat::cadcells::representation::util::IntervalCompare< T > Struct Template Reference

`#include <heuristics_covering.h>`

Public Member Functions

- constexpr bool operator() (const datastructures::SampledDerivationRef< T > &a, const datastructures::SampledDerivationRef< T > &b) const

0.15.251.1 Member Function Documentation

0.15.251.1.1 operator()() `template<typename T >`

```
constexpr bool smtrat::cadcells::representation::util::IntervalCompare< T >::operator() (
            const datastructures::SampledDerivationRef< T > & a,
            const datastructures::SampledDerivationRef< T > & b ) const [inline], [constexpr]
```

0.15.252 smtrat::covering_ng::IntervalCompare< op > Struct Template Reference

Sorts interval by their lower bounds.

`#include <types.h>`

Public Member Functions

- constexpr bool operator() (const Interval< op > &a, const Interval< op > &b) const

0.15.252.1 Detailed Description

template<cadcells::operators::op op>
struct smtrat::covering_ng::IntervalCompare< op >

Sorts interval by their lower bounds.

0.15.252.2 Member Function Documentation

0.15.252.2.1 operator()()

```
template<cadcells::operators::op op>
constexpr bool smtrat::covering_ng::IntervalCompare< op >::operator() (
            const Interval< op > & a,
            const Interval< op > & b ) const  [inline], [constexpr]
```

0.15.253 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 ()

0.15.253.1 Constructor & Destructor Documentation

0.15.253.1.1 IntervalPropagation()

```
smtrat::mcsat::icp::IntervalPropagation::IntervalPropagation (
            const std::vector< carl::Variable > & vars,
            const std::vector< FormulaT > & constraints,
            const Model & model )  [inline]
```

0.15.253.2 Member Function Documentation

0.15.253.2.1 execute()

```
std::optional<FormulaT> smtrat::mcsat::icp::IntervalPropagation::execute
( )  [inline]
```

0.15.254 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

0.15.254.1 Constructor & Destructor Documentation

0.15.254.1.1 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]
```

0.15.254.2 Member Function Documentation

0.15.254.2.1 getHelpPrefixString()

```
static const char*& Minisat::Option::getHelpPrefixString ( )
[inline], [static], [protected], [inherited]
```

0.15.254.2.2 getOptionList()

```
static vec<Option*>& Minisat::Option::getOptionList ( )  [inline],
[static], [protected], [inherited]
```

0.15.254.2.3 getUsageString()

```
static const char*& Minisat::Option::getUsageString ( )  [inline],
[static], [protected], [inherited]
```

0.15.254.2.4 help()

```
virtual void Minisat::IntOption::help (
    bool verbose = false )  [inline], [virtual]
```

Implements Minisat::Option.

0.15.254.2.5 operator int32_t()

```
Minisat::IntOption::operator int32_t (
    void  ) const  [inline]
```

0.15.254.2.6 operator=()

```
IntOption& Minisat::IntOption::operator= (
    int32_t x )  [inline]
```

0.15.254.3 Field Documentation

0.15.254.3.1 category

```
const char* Minisat::Option::category  [protected], [inherited]
```

0.15.254.3.2 description

```
const char* Minisat::Option::description  [protected], [inherited]
```

0.15.254.3.3 name `const char* Minisat::Option::name [protected], [inherited]`

0.15.254.3.4 range `IntRange Minisat::IntOption::range [protected]`

0.15.254.3.5 type_name `const char* Minisat::Option::type_name [protected], [inherited]`

0.15.254.3.6 value `int32_t Minisat::IntOption::value [protected]`

0.15.255 Minisat::IntRange Struct Reference

`#include <Options.h>`

Public Member Functions

- IntRange (int b, int e)

Data Fields

- int begin
- int end

0.15.255.1 Constructor & Destructor Documentation

0.15.255.1.1 IntRange()

```
Minisat::IntRange::IntRange (
    int b,
    int e ) [inline]
```

0.15.255.2 Field Documentation

0.15.255.2.1 begin `int Minisat::IntRange::begin`

0.15.255.2.2 end `int Minisat::IntRange::end`

0.15.256 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)

0.15.256.1 Member Function Documentation

0.15.256.1.1 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]
```

0.15.257 smtrat::is_sample_outside< IsSampleOutsideAlgorithm::DEFAULT > Struct Reference

`#include <Sampling.h>`

Static Public Member Functions

- template<typename T >
  static bool is_outside (const cadcells::RAN &sample, const std::set< cadcells::datastructures::SampledDerivationRef< T >, SampledDerivationRefCompare > &derivations)

0.15.257.1 Member Function Documentation

0.15.257.1.1 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]
```

0.15.258 smtrat::is_variant< T > Struct Template Reference

States whether a given type is a `boost::variant`.
`#include <VariantMap.h>`

0.15.258.1 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.

0.15.259 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

0.15.259.1 Constructor & Destructor Documentation

0.15.259.1.1 ITEExpression()

```
smtrat::expression::ITEExpression::ITEExpression (
    ITEType ,
    Expression && _condition,
    Expression && _then,
    Expression && _else )  [inline]
```

0.15.259.2 Field Documentation

0.15.259.2.1 condition

```
Expression smtrat::expression::ITEExpression::condition
```

0.15.259.2.2 elsecase

```
Expression smtrat::expression::ITEExpression::elsecase
```

0.15.259.2.3 thencase

```
Expression smtrat::expression::ITEExpression::thencase
```

0.15.259.2.4 type

```
ITEType smtrat::expression::ITEExpression::type
```

0.15.260 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.

0.15.260.1 Constructor & Destructor Documentation

0.15.260.1.1 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.

0.15.260.2 Member Function Documentation

0.15.260.2.1 operator"!=()

```
template<typename T >
bool benchmax::RandomizationAdaptor< T >::iterator::operator!= (
    const iterator & rhs ) const  [inline]
```

Compare two iterators.

0.15.260.2.2 operator*() `template<typename T >`

`const T& benchmax::RandomizationAdaptor< T >::iterator::operator* ( ) [inline]`

Dereference iterator.

0.15.260.2.3 operator++() `template<typename T >`

`iterator& benchmax::RandomizationAdaptor< T >::iterator::operator++ ( ) [inline]`

Increment iterator.

0.15.261 smtrat::ModuleInput::IteratorCompare Struct Reference

`#include <ModuleInput.h>`

Public Member Functions

- bool operator() (const_iterator i1, const_iterator i2) const

0.15.261.1 Member Function Documentation

0.15.261.1.1 operator()() `bool smtrat::ModuleInput::IteratorCompare::operator() ( const_iterator i1, const_iterator i2 ) const [inline]`

0.15.262 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.

0.15.262.1 Detailed Description

Represents a set of jobs, constructed as the cartesian product of a set of tools and a set of benchmarks.

0.15.262.2 Constructor & Destructor Documentation

0.15.262.2.1 Jobs()

```
benchmax::Jobs::Jobs (
    const Tools & tools,
    const BenchmarkSet & benchmarks ) [inline]
```

Construct jobs from a set of tools and a set of benchmark files.

0.15.262.3 Member Function Documentation

0.15.262.3.1 begin()

```
auto benchmax::Jobs::begin ( ) const [inline]
```

Begin iterator.

0.15.262.3.2 end()

```
auto benchmax::Jobs::end ( ) const [inline]
```

End iterator.

0.15.262.3.3 files()

```
const auto& benchmax::Jobs::files ( ) const [inline]
```

Returns the set of files.

0.15.262.3.4 randomized()

```
auto benchmax::Jobs::randomized ( ) const [inline]
```

Returns all jobs in a pseudo-randomized order.

0.15.262.3.5 size()

```
auto benchmax::Jobs::size ( ) const [inline]
```

Returns the overall number of jobs.

0.15.262.3.6 tools()

```
const auto& benchmax::Jobs::tools ( ) const [inline]
```

Returns the set of tools.

0.15.263 smtrat::parser::KeywordParser Struct Reference

Parses keywords: `:simple_symbol`

```
#include <Lexicon.h>
```

Public Member Functions

- KeywordParser ()

Data Fields

- qi::rule< Iterator, std::string(), Skipper > main
- SimpleSymbolParser simple

0.15.263.1 Detailed Description

Parses keywords: `:simple_symbol`

0.15.263.2 Constructor & Destructor Documentation

0.15.263.2.1 KeywordParser()

```
smtrat::parser::KeywordParser::KeywordParser ( ) [inline]
```

0.15.263.3 Field Documentation

0.15.263.3.1 main `qi::rule<Iterator, std::string(), Skipper> smtrat::parser::KeywordParser↩ ::main`

0.15.263.3.2 simple `SimpleSymbolParser smtrat::parser::KeywordParser::simple`

0.15.264 Minisat::lbool Class Reference

`#include <SolverTypes.h>`

Public Member Functions

- lbool (uint8_t v)
- lbool ()
- lbool (bool x)
- bool operator== (lbool b) const
- bool operator!= (lbool b) const
- lbool operator^ (bool b) const
- lbool operator&& (lbool b) const
- lbool operator|| (lbool b) const

Friends

- int toInt (lbool l)
- lbool toLbool (int v)

0.15.264.1 Constructor & Destructor Documentation

0.15.264.1.1 lbool() [1/3] `Minisat::lbool::lbool ( uint8_t v ) [inline], [explicit]`

0.15.264.1.2 lbool() [2/3] `Minisat::lbool::lbool ( ) [inline]`

0.15.264.1.3 lbool() [3/3] `Minisat::lbool::lbool ( bool x ) [inline], [explicit]`

0.15.264.2 Member Function Documentation

0.15.264.2.1 operator"!=() `bool Minisat::lbool::operator!= ( lbool b ) const [inline]`

0.15.264.2.2 operator&&() `lbool Minisat::lbool::operator&& ( lbool b ) const [inline]`

0.15.264.2.3 operator==()

```
bool Minisat::lbool::operator== (
lbool b ) const [inline]
```

0.15.264.2.4 operator^()

```
lbool Minisat::lbool::operator^ (
bool b ) const [inline]
```

0.15.264.2.5 operator"|"|()

```
lbool Minisat::lbool::operator|| (
lbool b ) const [inline]
```

0.15.264.3 Friends And Related Function Documentation

0.15.264.3.1 toInt

```
int toInt (
lbool l ) [friend]
```

0.15.264.3.2 toLbool

```
lbool toLbool (
int v ) [friend]
```

0.15.265 smtrat::mcsat::onecell::LDBFilteredAllSelectiveSettings Struct Reference

```
#include <onecell.h>
```

Static Public Attributes

- constexpr static auto cell_heuristic = cadcells::representation::LOWEST_DEGREE_BARRIERS_EW
- constexpr static auto covering_heuristic = cadcells::representation::BIGGEST_CELL_COVERING_EW
- constexpr static auto op = cadcells::operators::op::mccallum_filtered_all_selective
- constexpr static bool use_approximation = false

0.15.265.1 Field Documentation

0.15.265.1.1 cell_heuristic

```
constexpr static auto smtrat::mcsat::onecell::LDBFilteredAllSelective↩
Settings::cell_heuristic = cadcells::representation::LOWEST_DEGREE_BARRIERS_EW [static],
[constexpr]
```

0.15.265.1.2 covering_heuristic

```
constexpr static auto smtrat::mcsat::onecell::LDBFilteredAll↩
SelectiveSettings::covering_heuristic = cadcells::representation::BIGGEST_CELL_COVERING_EW
[static], [constexpr]
```

0.15.265.1.3 op

```
constexpr static auto smtrat::mcsat::onecell::LDBFilteredAllSelectiveSettings↩
::op = cadcells::operators::op::mccallum_filtered_all_selective [static], [constexpr]
```

0.15.265.1.4 use_approximation

```
constexpr static bool smtrat::mcsat::onecell::Base::use_↩
approximation = false [static], [constexpr], [inherited]
```

0.15.266 smtrat::mcsat::onecell::LDBSettings Struct Reference

```
#include <onecell.h>
```

Static Public Attributes

- constexpr static auto cell_heuristic = cadcells::representation::LOWEST_DEGREE_BARRIERS
- constexpr static auto covering_heuristic = cadcells::representation::BIGGEST_CELL_COVERING
- constexpr static auto op = cadcells::operators::op::mccallum
- constexpr static bool use_approximation = false

0.15.266.1 Field Documentation

0.15.266.1.1 cell_heuristic

`constexpr static auto smtrat::mcsat::onecell::LDBSettings::cell_heuristic = cadcells::representation::LOWEST_DEGREE_BARRIERS [static], [constexpr]`

0.15.266.1.2 covering_heuristic

`constexpr static auto smtrat::mcsat::onecell::LDBSettings::covering_heuristic = cadcells::representation::BIGGEST_CELL_COVERING [static], [constexpr]`

0.15.266.1.3 op

`constexpr static auto smtrat::mcsat::onecell::LDBSettings::op = cadcells::operators::op::mccallum [static], [constexpr]`

0.15.266.1.4 use_approximation

`constexpr static bool smtrat::mcsat::onecell::Base::use_approximation = false [static], [constexpr], [inherited]`

0.15.267 smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound Struct Reference

`#include <Tableau.h>`

Public Member Functions

- LearnedBound ()=delete
- LearnedBound (const LearnedBound &)=delete
- LearnedBound (LearnedBound &&_toMove)
- LearnedBound (const Value< T1 > &_limit, const Bound< T1, T2 > *_newBound, typename Bound< T1, T2 >::BoundSet::const_iterator _nextWeakerBound, std::vector< const Bound< T1, T2 > * > &&_premise)
- ~LearnedBound ()

Data Fields

- const Bound< T1, T2 > * newBound
- Value< T1 > newLimit
- Bound< T1, T2 >::BoundSet::const_iterator nextWeakerBound
- std::vector< const Bound< T1, T2 > * > premise

0.15.267.1 Constructor & Destructor Documentation

0.15.267.1.1 LearnedBound() [1/4]

`template<class Settings , typename T1 , typename T2 > smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound::LearnedBound ( ) [delete]`

0.15.267.1.2 LearnedBound() [2/4]

`template<class Settings , typename T1 , typename T2 > smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound::LearnedBound ( const LearnedBound & ) [delete]`

0.15.267.1.3 LearnedBound() [3/4]

```
template<class Settings , typename T1 , typename T2 >
smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound::LearnedBound (
            LearnedBound && _toMove )  [inline]
```

0.15.267.1.4 LearnedBound() [4/4]

```
template<class Settings , typename T1 , typename T2 >
smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound::LearnedBound (
            const Value< T1 > & _limit,
            const Bound< T1, T2 > * _newBound,
            typename Bound< T1, T2 >::BoundSet::const_iterator _nextWeakerBound,
            std::vector< const Bound< T1, T2 > * > && _premise )  [inline]
```

0.15.267.1.5 ~LearnedBound()

```
template<class Settings , typename T1 , typename T2 >
smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound::~LearnedBound ( )  [inline]
```

0.15.267.2 Field Documentation

0.15.267.2.1 newBound

```
template<class Settings , typename T1 , typename T2 >
const Bound<T1, T2>* smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound::newBound
```

0.15.267.2.2 newLimit

```
template<class Settings , typename T1 , typename T2 >
Value<T1> smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound::newLimit
```

0.15.267.2.3 nextWeakerBound

```
template<class Settings , typename T1 , typename T2 >
Bound<T1, T2>::BoundSet::const_iterator smtrat::lra::Tableau< Settings, T1, T2 >::Learned↩
Bound::nextWeakerBound
```

0.15.267.2.4 premise

```
template<class Settings , typename T1 , typename T2 >
std::vector< const Bound<T1, T2>*> smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound↩
::premise
```

0.15.268 smtrat::Module::Lemma Struct Reference

```
#include <Module.h>
```

Public Member Functions

- Lemma (const FormulaT &_lemma, const LemmaType &_lemmaType, const FormulaT &_preferredFormula)

  Constructor.

Data Fields

- FormulaT mLemma

  The lemma to learn.
- LemmaType mLemmaType

  The type of the lemma.
- FormulaT mPreferredFormula

  The formula within the lemma, which should be assigned to true in the next decision.

0.15.268.1 Constructor & Destructor Documentation

0.15.268.1.1 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.

0.15.268.2 Field Documentation

0.15.268.2.1 mLemma

`FormulaT smtrat::Module::Lemma::mLemma`

The lemma to learn.

0.15.268.2.2 mLemmaType

`LemmaType smtrat::Module::Lemma::mLemmaType`

The type of the lemma.

0.15.268.2.3 mPreferredFormula

`FormulaT smtrat::Module::Lemma::mPreferredFormula`

The formula within the lemma, which should be assigned to true in the next decision.

0.15.269 Minisat::LessThan_default< T > Struct Template Reference

`#include <Sort.h>`

Public Member Functions

- bool operator() (T x, T y)

0.15.269.1 Member Function Documentation

0.15.269.1.1 operator()()

```
template<class T >
bool Minisat::LessThan_default< T >::operator() (
    T x,
    T y ) [inline]
```

0.15.270 smtrat::cad::projection_compare::level Struct Reference

`#include <ProjectionComparator.h>`

0.15.271 smtrat::cad::sample_compare::level Struct Reference

`#include <SampleComparator.h>`

0.15.272 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.

0.15.272.1 Member Function Documentation

0.15.272.1.1 isBound()

`bool smtrat::cad::ProjectionLevelInformation::LevelInfo::isBound ( std::size_t pid ) const [inline]`

0.15.272.1.2 isEvaluated()

`bool smtrat::cad::ProjectionLevelInformation::LevelInfo::isEvaluated ( std::size_t pid ) const [inline]`

0.15.272.1.3 isPurged()

`bool smtrat::cad::ProjectionLevelInformation::LevelInfo::isPurged ( std::size_t pid ) const [inline]`

0.15.272.1.4 removePurgedFromEvaluated()

`void smtrat::cad::ProjectionLevelInformation::LevelInfo::removePurgedFromEvaluated ( ) [inline]`

0.15.272.1.5 restrictEvaluatedToPurged()

`void smtrat::cad::ProjectionLevelInformation::LevelInfo::restrictEvaluatedToPurged ( ) [inline]`

0.15.272.1.6 setBound()

`void smtrat::cad::ProjectionLevelInformation::LevelInfo::setBound ( std::size_t pid, bool isBound ) [inline]`

0.15.272.1.7 setEvaluated()

```
void smtrat::cad::ProjectionLevelInformation::LevelInfo::setEvaluated (
    std::size_t pid,
    bool isEvaluated ) [inline]
```

0.15.272.1.8 setPurged()

```
void smtrat::cad::ProjectionLevelInformation::LevelInfo::setPurged (
    std::size_t pid,
    bool isPurged ) [inline]
```

0.15.272.2 Field Documentation

0.15.272.2.1 bounds

`carl::Bitset smtrat::cad::ProjectionLevelInformation::LevelInfo::bounds`

Which polynomials are bounds.

0.15.272.2.2 ecs

`EquationalConstraints smtrat::cad::ProjectionLevelInformation::LevelInfo::ecs`

Equational constraints.

0.15.272.2.3 evaluated

`carl::Bitset smtrat::cad::ProjectionLevelInformation::LevelInfo::evaluated`

Which polynomials have been evaluated w.r.t. purging.

0.15.272.2.4 purged

`carl::Bitset smtrat::cad::ProjectionLevelInformation::LevelInfo::purged`

Which polynomials are purged from the projection. (usually due to bounds)

0.15.273 smtrat::mcsat::onecellcad::levelwise::LevelwiseCAD Class Reference

`#include <OneCellCAD.h>`

Public Member Functions

- std::optional< CADCell > constructCADCellEnclosingPoint (std::vector< std::vector< TagPoly >> &polys, int sectionHeuristic, int sectorHeuristic)

  Construct a single CADCell that contains the given 'point' and is sign-invariant for the given polynomials in 'polys'.
- OneCellCAD (const std::vector< carl::Variable > &variableOrder, const RealAlgebraicPoint< Rational > &point)
- std::map< carl::Variable, RAN > prefixPointToStdMap (const std::size_t componentCount)

  Create a mapping from the first variables (with index 0 to 'componentCount') in the 'variableOrder' to the first components of the given 'point'.
- std::vector< RAN > isolateLastVariableRoots (const std::size_t polyLevel, const Poly &poly)

  src/lib/datastructures/mcsat/onecellcad/OneCellCAD.h Given a poly p(x1,..,xn), return all roots of the univariate poly p(a1,..,an-1, xn), where a1,..an-1 are the first algebraic real components from 'point' (SORTED!).
- bool vanishesEarly (const std::size_t polyLevel, const Poly &poly)

  Check if an n-variate 'poly' p(x1,..,xn) with n>=1 already becomes the zero poly after plugging in p(a1,..,an-1, xn), where a1,..an-1 are the first n-1 algebraic real components from 'point'.
- bool isPointRootOfPoly (const std::size_t polyLevel, const Poly &poly)
- bool isPointRootOfPoly (const TagPoly &poly)
- std::optional< Poly > coeffNonNull (const TagPoly &boundCandidate)
- bool isMainPointInsideCell (const CADCell &cell)

Data Fields

- const std::vector< carl::Variable > & variableOrder

  Variables can also be indexed by level.
- const RealAlgebraicPoint< Rational > & point

0.15.273.1 Member Function Documentation

0.15.273.1.1 coeffNonNull()

```
std::optional<Poly> smtrat::mcsat::onecellcad::OneCellCAD::coeff↩
NonNull (
            const TagPoly & boundCandidate )  [inline], [inherited]
```

0.15.273.1.2 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

0.15.273.1.3 isMainPointInsideCell()

```
bool smtrat::mcsat::onecellcad::OneCellCAD::isMainPoint↩
InsideCell (
            const CADCell & cell )  [inline], [inherited]
```

0.15.273.1.4 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!).

0.15.273.1.5 isPointRootOfPoly() [1/2]

```
bool smtrat::mcsat::onecellcad::OneCellCAD::isPointRoot↩
OfPoly (
            const std::size_t polyLevel,
            const Poly & poly )  [inline], [inherited]
```

0.15.273.1.6 isPointRootOfPoly() [2/2]

```
bool smtrat::mcsat::onecellcad::OneCellCAD::isPointRootOfPoly (
    const TagPoly & poly ) [inline], [inherited]
```

0.15.273.1.7 OneCellCAD()

```
smtrat::mcsat::onecellcad::OneCellCAD::OneCellCAD [inline]
```

0.15.273.1.8 prefixPointToStdMap()

```
std::map<carl::Variable, RAN> smtrat::mcsat::onecellcad::OneCellCAD::prefixPointToStdMap (
    const std::size_t componentCount ) [inline], [inherited]
```

Create a mapping from the first variables (with index 0 to 'componentCount') in the 'variableOrder' to the first components of the given 'point'.

0.15.273.1.9 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'.

0.15.273.2 Field Documentation

0.15.273.2.1 point

```
const RealAlgebraicPoint<Rational>& smtrat::mcsat::onecellcad::OneCellCAD::point [inherited]
```

0.15.273.2.2 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]

0.15.274 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 cadcells::RAN & getSampleOutside () const
- bool isPartialCovering () const
- bool isFullCovering () const
- bool isUnknownCovering () const
- bool isFailedCovering () const
- CoveringStatus getCoveringStatus () const
- LevelWiseInformation & operator= (LevelWiseInformation &&other)
- void setDerivations (std::vector< datastructures::SampledDerivationRef< PropSet >> &&derivations)

- void removeDerivation (const datastructures::SampledDerivationRef< PropSet > &derivation)
- void removeDerivation (const std::vector< datastructures::SampledDerivationRef< PropSet >> &derivations)
- void removeConstraint (const cadcells::Constraint &constraint, const std::map< datastructures::SampledDerivationRef< PropSet >, std::vector< cadcells::Constraint >> &derivationConstraints)
- std::vector< cadcells::Constraint > getConstraintsOfCovering (std::map< datastructures::SampledDerivationRef< PropSet >, std::vector< cadcells::Constraint >> &mDerivationToConstraint)
- std::optional< datastructures::SampledDerivationRef< PropSet > > constructDerivation (std::map< datastructures::SampledDerivationRef< PropSet >, std::vector< cadcells::Constraint >> &mDerivation↩ ToConstraint)

0.15.274.1 Constructor & Destructor Documentation

0.15.274.1.1 LevelWiseInformation() [1/2]

```
template<class Settings >
smtrat::LevelWiseInformation< Settings >::LevelWiseInformation ( ) [inline]
```

0.15.274.1.2 LevelWiseInformation() [2/2]

```
template<class Settings >
smtrat::LevelWiseInformation< Settings >::LevelWiseInformation (
            LevelWiseInformation< Settings > && other ) [inline]
```

0.15.274.2 Member Function Documentation

0.15.274.2.1 addDerivation() [1/2]

```
template<class Settings >
void smtrat::LevelWiseInformation< Settings >::addDerivation (
            const datastructures::SampledDerivationRef< PropSet > & derivation ) [inline]
```

0.15.274.2.2 addDerivation() [2/2]

```
template<class Settings >
void smtrat::LevelWiseInformation< Settings >::addDerivation (
            datastructures::SampledDerivationRef< PropSet > && derivation ) [inline]
```

0.15.274.2.3 clear()

```
template<class Settings >
void smtrat::LevelWiseInformation< Settings >::clear ( ) [inline]
```

0.15.274.2.4 computeCovering()

```
template<class Settings >
bool smtrat::LevelWiseInformation< Settings >::computeCovering ( ) [inline]
```

0.15.274.2.5 constructDerivation()

```
template<class Settings >
std::optional<datastructures::SampledDerivationRef<PropSet> > smtrat::LevelWiseInformation<
Settings >::constructDerivation (
            std::map< datastructures::SampledDerivationRef< PropSet >, std::vector< cadcells::Constraint
>> & mDerivationToConstraint ) [inline]
```

0.15.274.2.6 getConstraintsOfCovering()

```
template<class Settings >
std::vector<cadcells::Constraint> smtrat::LevelWiseInformation< Settings >::getConstraints↩
OfCovering (
            std::map< datastructures::SampledDerivationRef< PropSet >, std::vector< cadcells::Constraint
>> & mDerivationToConstraint )  [inline]
```

0.15.274.2.7 getCoveringStatus()

```
template<class Settings >
CoveringStatus smtrat::LevelWiseInformation< Settings >::getCoveringStatus ( ) const  [inline]
```

0.15.274.2.8 getDerivations()

```
template<class Settings >
const std::set<datastructures::SampledDerivationRef<PropSet>, SampledDerivationRefCompare>&
smtrat::LevelWiseInformation< Settings >::getDerivations ( ) const  [inline]
```

0.15.274.2.9 getSampleOutside()

```
template<class Settings >
const cadcells::RAN& smtrat::LevelWiseInformation< Settings >::getSampleOutside ( ) const
[inline]
```

0.15.274.2.10 isFailedCovering()

```
template<class Settings >
bool smtrat::LevelWiseInformation< Settings >::isFailedCovering ( ) const  [inline]
```

0.15.274.2.11 isFullCovering()

```
template<class Settings >
bool smtrat::LevelWiseInformation< Settings >::isFullCovering ( ) const  [inline]
```

0.15.274.2.12 isPartialCovering()

```
template<class Settings >
bool smtrat::LevelWiseInformation< Settings >::isPartialCovering ( ) const  [inline]
```

0.15.274.2.13 isUnknownCovering()

```
template<class Settings >
bool smtrat::LevelWiseInformation< Settings >::isUnknownCovering ( ) const  [inline]
```

0.15.274.2.14 operator=()

```
template<class Settings >
LevelWiseInformation& smtrat::LevelWiseInformation< Settings >::operator= (
            LevelWiseInformation< Settings > && other )  [inline]
```

0.15.274.2.15 removeConstraint()

```
template<class Settings >
void smtrat::LevelWiseInformation< Settings >::removeConstraint (
            const cadcells::Constraint & constraint,
            const std::map< datastructures::SampledDerivationRef< PropSet >, std::vector<
cadcells::Constraint >> & derivationConstraints )  [inline]
```

0.15.274.2.16 removeDerivation() [1/2]

```
template<class Settings >
void smtrat::LevelWiseInformation< Settings >::removeDerivation (
            const datastructures::SampledDerivationRef< PropSet > & derivation )  [inline]
```

0.15.274.2.17 removeDerivation() [2/2] template<class Settings >

```
void smtrat::LevelWiseInformation< Settings >::removeDerivation (
            const std::vector< datastructures::SampledDerivationRef< PropSet >> & derivations
) [inline]
```

0.15.274.2.18 setDerivations() template<class Settings >

```
void smtrat::LevelWiseInformation< Settings >::setDerivations (
            std::vector< datastructures::SampledDerivationRef< PropSet >> && derivations )
[inline]
```

0.15.275 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 ignore_nullifications)
- bool addTrivialSample (Iterator sample)
- Assignment extractSampleMap (Iterator it) const
- void removedPolynomialsFromLevel (std::size_t level, const carl::Bitset &mask)
- void removedConstraint (const carl::Bitset &mask)
- bool is_consistent () const
- std::string printSample (Iterator sample) const
- std::string printFullSamples () const

0.15.275.1 Member Typedef Documentation

0.15.275.1.1 Constraints template<typename Settings >

```
using smtrat::cad::LiftingTree< Settings >::Constraints = CADConstraints<Settings::backtracking>
```

0.15.275.1.2 FSC

```
template<typename Settings >
using smtrat::cad::LiftingTree< Settings >::FSC = FullSampleComparator<Iterator,Settings↩
::fullSampleComparator>
```

0.15.275.1.3 Iterator

```
template<typename Settings >
using smtrat::cad::LiftingTree< Settings >::Iterator = Tree::iterator
```

0.15.275.1.4 SC

```
template<typename Settings >
using smtrat::cad::LiftingTree< Settings >::SC = SampleComparator<Iterator,Settings::sample↩
Comparator>
```

0.15.275.1.5 Tree

```
template<typename Settings >
using smtrat::cad::LiftingTree< Settings >::Tree = carl::tree<Sample>
```

0.15.275.2 Constructor & Destructor Documentation

0.15.275.2.1 LiftingTree() [1/3]

```
template<typename Settings >
smtrat::cad::LiftingTree< Settings >::LiftingTree (
            const Constraints & c )  [inline]
```

0.15.275.2.2 LiftingTree() [2/3]

```
template<typename Settings >
smtrat::cad::LiftingTree< Settings >::LiftingTree (
            const LiftingTree< Settings > &  )  [delete]
```

0.15.275.2.3 LiftingTree() [3/3]

```
template<typename Settings >
smtrat::cad::LiftingTree< Settings >::LiftingTree (
            LiftingTree< Settings > &&  )  [delete]
```

0.15.275.3 Member Function Documentation

0.15.275.3.1 addTrivialSample()

```
template<typename Settings >
bool smtrat::cad::LiftingTree< Settings >::addTrivialSample (
            Iterator sample )  [inline]
```

0.15.275.3.2 extractSampleMap()

```
template<typename Settings >
Assignment smtrat::cad::LiftingTree< Settings >::extractSampleMap (
            Iterator it ) const  [inline]
```

0.15.275.3.3 getLiftingQueue()

```
template<typename Settings >
const auto& smtrat::cad::LiftingTree< Settings >::getLiftingQueue ( ) const  [inline]
```

0.15.275.3.4 getNextFullSample()

```
template<typename Settings >
Iterator smtrat::cad::LiftingTree< Settings >::getNextFullSample ( )  [inline]
```

0.15.275.3.5 getNextSample()

```
template<typename Settings >
Iterator smtrat::cad::LiftingTree< Settings >::getNextSample ( ) [inline]
```

0.15.275.3.6 getTree()

```
template<typename Settings >
const auto& smtrat::cad::LiftingTree< Settings >::getTree ( ) const [inline]
```

0.15.275.3.7 hasFullSamples()

```
template<typename Settings >
bool smtrat::cad::LiftingTree< Settings >::hasFullSamples ( ) const [inline]
```

0.15.275.3.8 hasNextSample()

```
template<typename Settings >
bool smtrat::cad::LiftingTree< Settings >::hasNextSample ( ) const [inline]
```

0.15.275.3.9 is_consistent()

```
template<typename Settings >
bool smtrat::cad::LiftingTree< Settings >::is_consistent ( ) const [inline]
```

0.15.275.3.10 liftSample()

```
template<typename Settings >
bool smtrat::cad::LiftingTree< Settings >::liftSample (
            Iterator sample,
            const UPoly & p,
            std::size_t pid,
            bool ignore_nullifications ) [inline]
```

0.15.275.3.11 operator=() [1/2]

```
template<typename Settings >
LiftingTree& smtrat::cad::LiftingTree< Settings >::operator= (
            const LiftingTree< Settings > & ) [delete]
```

0.15.275.3.12 operator=() [2/2]

```
template<typename Settings >
LiftingTree& smtrat::cad::LiftingTree< Settings >::operator= (
            LiftingTree< Settings > && ) [delete]
```

0.15.275.3.13 printFullSamples()

```
template<typename Settings >
std::string smtrat::cad::LiftingTree< Settings >::printFullSamples ( ) const [inline]
```

0.15.275.3.14 printSample()

```
template<typename Settings >
std::string smtrat::cad::LiftingTree< Settings >::printSample (
            Iterator sample ) const [inline]
```

0.15.275.3.15 removedConstraint()

```
template<typename Settings >
void smtrat::cad::LiftingTree< Settings >::removedConstraint (
            const carl::Bitset & mask ) [inline]
```

0.15.275.3.16 removedPolynomialsFromLevel()

```
template<typename Settings >
void smtrat::cad::LiftingTree< Settings >::removedPolynomialsFromLevel (
            std::size_t level,
            const carl::Bitset & mask ) [inline]
```

0.15.275.3.17 removeNextSample()

```
template<typename Settings >
void smtrat::cad::LiftingTree< Settings >::removeNextSample ( ) [inline]
```

0.15.275.3.18 reset()

```
template<typename Settings >
void smtrat::cad::LiftingTree< Settings >::reset (
            std::vector< carl::Variable > && vars ) [inline]
```

0.15.275.3.19 resetFullSamples()

```
template<typename Settings >
void smtrat::cad::LiftingTree< Settings >::resetFullSamples ( ) [inline]
```

0.15.275.3.20 restoreRemovedSamples()

```
template<typename Settings >
void smtrat::cad::LiftingTree< Settings >::restoreRemovedSamples ( ) [inline]
```

0.15.276 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

0.15.276.1 Member Function Documentation

0.15.276.1.1 initialize()

```
void smtrat::resource::Limiter::initialize ( ) [inline]
```

0.15.276.1.2 reset()

```
void smtrat::resource::Limiter::reset ( ) [inline]
```

0.15.276.1.3 resetTimeout()

```
void smtrat::resource::Limiter::resetTimeout ( ) const [inline]
```

0.15.276.1.4 setMemout()

```
void smtrat::resource::Limiter::setMemout (
            std::size_t megabytes ) [inline]
```

0.15.276.1.5 setTimeout()

```
void smtrat::resource::Limiter::setTimeout (
            std::size_t seconds ) [inline]
```

0.15.276.1.6 setTimeoutHandler()

```
void smtrat::resource::Limiter::setTimeoutHandler (
    std::function< void()> f ) [inline]
```

0.15.276.1.7 timeoutHandler()

```
std::function<void()> smtrat::resource::Limiter::timeoutHandler (
) const [inline]
```

0.15.277 smtrat::ICPModule< Settings >::linearVariable Struct Reference

```
#include <ICPModule.h>
```

Data Fields

- FormulaT constraint
- ConstraintT origin

0.15.277.1 Field Documentation

0.15.277.1.1 constraint

```
template<class Settings >
FormulaT smtrat::ICPModule< Settings >::linearVariable::constraint
```

0.15.277.1.2 origin

```
template<class Settings >
ConstraintT smtrat::ICPModule< Settings >::linearVariable::origin
```

0.15.278 smtrat::mcsat::ClauseChain::Link Struct Reference

```
#include <ClauseChain.h>
```

Public Member Functions

- Link (const FormulaT &&clause, const FormulaT &&impliedTseitinLiteral)
- Link (const FormulaT &&clause)
- bool isPropagating () const
- bool isConflicting () const
- bool isOptional () const
- const FormulaT & clause () const
- const FormulaT & impliedTseitinLiteral () const

Friends

- std::ostream & operator<< (std::ostream &stream, const Link &link)

0.15.278.1 Constructor & Destructor Documentation

0.15.278.1.1 Link() [1/2]

```
smtrat::mcsat::ClauseChain::Link::Link (
    const FormulaT && clause,
    const FormulaT && impliedTseitinLiteral ) [inline]
```

0.15.278.1.2 Link() [2/2]

```
smtrat::mcsat::ClauseChain::Link::Link (
    const FormulaT && clause ) [inline]
```

0.15.278.2 Member Function Documentation

0.15.278.2.1 clause() `const FormulaT& smtrat::mcsat::ClauseChain::Link::clause ( ) const [inline]`

0.15.278.2.2 impliedTseitinLiteral() `const FormulaT& smtrat::mcsat::ClauseChain::Link::implied↩ TseitinLiteral ( ) const [inline]`

0.15.278.2.3 isConflicting() `bool smtrat::mcsat::ClauseChain::Link::isConflicting ( ) const [inline]`

0.15.278.2.4 isOptional() `bool smtrat::mcsat::ClauseChain::Link::isOptional ( ) const [inline]`

0.15.278.2.5 isPropagating() `bool smtrat::mcsat::ClauseChain::Link::isPropagating ( ) const [inline]`

0.15.278.3 Friends And Related Function Documentation

0.15.278.3.1 operator<< `std::ostream& operator<< ( std::ostream & stream, const Link & link ) [friend]`

0.15.279 Minisat::Lit Struct Reference

`#include <SolverTypes.h>`

Public Member Functions

- bool operator== (Lit p) const
- bool operator!= (Lit p) const
- bool operator< (Lit p) const

Data Fields

- int x

Friends

- Lit mkLit (Var var, bool sign)

0.15.279.1 Member Function Documentation

0.15.279.1.1 operator"!=() `bool Minisat::Lit::operator!= ( Lit p ) const [inline]`

0.15.279.1.2 operator<() `bool Minisat::Lit::operator< ( Lit p ) const [inline]`

0.15.279.1.3 operator==()

```
bool Minisat::Lit::operator== (
Lit p ) const [inline]
```

0.15.279.2 Friends And Related Function Documentation

0.15.279.2.1 mkLit

```
Lit mkLit (
Var var,
bool sign = false ) [friend]
```

0.15.279.3 Field Documentation

0.15.279.3.1 x

```
int Minisat::Lit::x
```

0.15.280 benchmax::LocalBackend Class Reference

This backend simply runs files sequentially on the local machine.

`#include <LocalBackend.h>`

Public Member Functions

- bool suspendable () const
- void process_results (const Jobs &jobs, bool check_finished)
- void addResult (const Tool *tool, const fs::path &file, BenchmarkResult &&result)

  Add a result.
- void run (const Jobs &jobs, bool wait_for_termination)

  Run the list of tools against the list of benchmarks.

Protected Member Functions

- virtual void execute (const Tool *tool, const fs::path &file)

  Execute the tool on the file manually.
- virtual void startTool (const Tool *)

  Hook for every tool at the beginning.
- virtual void finalize ()

  Hook to allow for asynchronous backends to wait for jobs to terminate.
- void madeProgress (std::size_t files=1)

  Can be called to give information about the current progress, if available.
- virtual bool collect_results (const Jobs &, bool)
- void sanitize_results (const Jobs &jobs) const
- void write_results (const Jobs &jobs) const

Protected Attributes

- std::size_t mExpectedJobs

  Number of jobs that should be run.
- std::atomic< std::size_t > mFinishedJobs

  Number of jobs that are finished.
- std::atomic< std::size_t > mLastPercent

  Percentage of finished jobs when madeProgress() was last called.

0.15.280.1 Detailed Description

This backend simply runs files sequentially on the local machine.

0.15.280.2 Member Function Documentation

0.15.280.2.1 addResult()

```
void benchmax::Backend::addResult (
    const Tool * tool,
    const fs::path & file,
    BenchmarkResult && result ) [inline], [inherited]
```

Add a result.

0.15.280.2.2 collect_results()

```
virtual bool benchmax::Backend::collect_results (
    const Jobs & ,
    bool ) [inline], [protected], [virtual], [inherited]
```

0.15.280.2.3 execute()

```
virtual void benchmax::LocalBackend::execute (
    const Tool * tool,
    const fs::path & file ) [inline], [protected], [virtual]
```

Execute the tool on the file manually.
Reimplemented from benchmax::Backend.

0.15.280.2.4 finalize()

```
virtual void benchmax::Backend::finalize ( ) [inline], [protected],
[virtual], [inherited]
```

Hook to allow for asynchronous backends to wait for jobs to terminate.

0.15.280.2.5 madeProgress()

```
void benchmax::Backend::madeProgress (
    std::size_t files = 1 ) [inline], [protected], [inherited]
```

Can be called to give information about the current progress, if available.

0.15.280.2.6 process_results()

```
void benchmax::Backend::process_results (
    const Jobs & jobs,
    bool check_finished ) [inline], [inherited]
```

0.15.280.2.7 run()

```
void benchmax::Backend::run (
    const Jobs & jobs,
    bool wait_for_termination ) [inline], [inherited]
```

Run the list of tools against the list of benchmarks.

0.15.280.2.8 sanitize_results()

```
void benchmax::Backend::sanitize_results (
    const Jobs & jobs ) const [inline], [protected], [inherited]
```

0.15.280.2.9 startTool()

```
virtual void benchmax::Backend::startTool (
    const Tool * ) [inline], [protected], [virtual], [inherited]
```

Hook for every tool at the beginning.
Can be used to upload the tool to some remote system.

0.15.280.2.10 suspendable() `bool benchmax::Backend::suspendable ( ) const [inline], [inherited]`

0.15.280.2.11 write_results() `void benchmax::Backend::write_results ( const Jobs & jobs ) const [inline], [protected], [inherited]`

0.15.280.3 Field Documentation

0.15.280.3.1 mExpectedJobs `std::size_t benchmax::Backend::mExpectedJobs [protected], [inherited]`

Number of jobs that should be run.

0.15.280.3.2 mFinishedJobs `std::atomic<std::size_t> benchmax::Backend::mFinishedJobs [protected], [inherited]`

Number of jobs that are finished.

0.15.280.3.3 mLastPercent `std::atomic<std::size_t> benchmax::Backend::mLastPercent [protected], [inherited]`

Percentage of finished jobs when madeProgress() was last called.

0.15.281 smtrat::LOG Class Reference

`#include <LOG.h>`

Public Member Functions

- bool isDebugEnabled ()

Data Fields

- bool debugEnabled = false

Friends

- class carl::Singleton< LOG >

0.15.281.1 Member Function Documentation

0.15.281.1.1 isDebugEnabled() `bool smtrat::LOG::isDebugEnabled ( ) [inline]`

0.15.281.2 Friends And Related Function Documentation

0.15.281.2.1 carl::Singleton< LOG > `friend class carl::Singleton< LOG > [friend]`

0.15.281.3 Field Documentation

0.15.281.3.1 debugEnabled `bool smtrat::LOG::debugEnabled = false`

0.15.282 smtrat::parser::LogicParser Struct Reference

`#include <Script.h>`

Public Member Functions

- LogicParser ()

0.15.282.1 Constructor & Destructor Documentation

0.15.282.1.1 LogicParser() `smtrat::parser::LogicParser::LogicParser ( ) [inline]`

0.15.283 smtrat::LongFormulaEncoder Class Reference

`#include <LongFormulaEncoder.h>`

Public Member Functions

- LongFormulaEncoder ()
- bool canEncode (const ConstraintT &constraint)
- Rational encodingSize (const ConstraintT &constraint)
- std::string name ()
- std::optional< FormulaT > encode (const ConstraintT &constraint)

  Encodes an arbitrary constraint.

Data Fields

- std::size_t problem_size

Protected Member Functions

- std::optional< FormulaT > doEncode (const ConstraintT &constraint)
- FormulaT generateVarChain (const std::set< carl::Variable > &vars, carl::FormulaType type)

0.15.283.1 Constructor & Destructor Documentation

0.15.283.1.1 LongFormulaEncoder() `smtrat::LongFormulaEncoder::LongFormulaEncoder ( ) [inline]`

0.15.283.2 Member Function Documentation

0.15.283.2.1 canEncode() `bool smtrat::LongFormulaEncoder::canEncode ( const ConstraintT & constraint ) [virtual]`

Implements smtrat::PseudoBoolEncoder.

0.15.283.2.2 doEncode() `std::optional< FormulaT > smtrat::LongFormulaEncoder::doEncode ( const ConstraintT & constraint ) [protected], [virtual]`

Implements smtrat::PseudoBoolEncoder.

0.15.283.2.3 encode() `std::optional< FormulaT > smtrat::PseudoBoolEncoder::encode ( const ConstraintT & constraint ) [inherited]`

Encodes an arbitrary constraint.

Returns

encoded formula

0.15.283.2.4 encodingSize() `Rational smtrat::LongFormulaEncoder::encodingSize ( const ConstraintT & constraint ) [virtual]`

Reimplemented from smtrat::PseudoBoolEncoder.

0.15.283.2.5 generateVarChain() `FormulaT smtrat::PseudoBoolEncoder::generateVarChain ( const std::set< carl::Variable > & vars, carl::FormulaType type ) [protected], [inherited]`

0.15.283.2.6 name() `std::string smtrat::LongFormulaEncoder::name ( ) [inline], [virtual]`

Reimplemented from smtrat::PseudoBoolEncoder.

0.15.283.3 Field Documentation

0.15.283.3.1 problem_size `std::size_t smtrat::PseudoBoolEncoder::problem_size [inherited]`

0.15.284 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 class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- LRAModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=NULL)

  Constructs a LRAModule.
- virtual ~LRAModule ()

  Destructs this LRAModule.
- bool informCore (const FormulaT &_constraint)

  Informs this module about the existence of the given constraint, which means that it could be added in future.
- void deinformCore (const FormulaT &_constraint)

  The inverse of informing about a constraint.
- void init ()

  Initializes the tableau according to all linear constraints, of which this module has been informed.
- bool addCore (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- void removeCore (ModuleInput::const_iterator _subformula)

  Removes everything related to the given sub-formula of the received formula.
- Answer checkCore ()

  Checks the received formula for consistency.
- Answer checkCore_old ()
- Answer processResult (Answer _result)
- void updateModel () const

  Updates the model, if the solver has detected the consistency of the received formula, beforehand.
- unsigned currentlySatisfied (const FormulaT &) const
- const RationalAssignment & getRationalModel () const

  Gives a rational model if the received formula is satisfiable.
- Answer optimize (Answer _result)
- Answer checkNotEqualConstraints (Answer _result)
- void processLearnedBounds ()
- void createInfeasibleSubsets (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

- virtual bool informCore ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.
- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)
- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

Copies the infeasible subsets of the passed formula.

- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.284.1 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.

0.15.284.2 Member Typedef Documentation

0.15.284.2.1 ConstraintBoundsMap `template<class Settings >`

`typedef carl::FastMap<FormulaT, std::vector<const LRABound*>*> smtrat::LRAModule< Settings >::ConstraintBoundsMap`

Maps constraint to the bounds it represents (e.g., equations represent two bounds)

0.15.284.2.2 ConstraintContextMap `template<class Settings >`

`typedef carl::FastMap<FormulaT, Context> smtrat::LRAModule< Settings >::ConstraintContextMap`

Stores the position of a received formula in the passed formula.

0.15.284.2.3 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.

0.15.284.2.4 LRABound `template<class Settings >`

`typedef lra::Bound<LRABoundType, LRAEntryType> smtrat::LRAModule< Settings >::LRABound`

Type of a bound of a variable within the LRAModule.

0.15.284.2.5 LRABoundType `template<class Settings >`

`typedef Settings::BoundType smtrat::LRAModule< Settings >::LRABoundType`

Number type of the underlying value of a bound of a variable within the LRAModule.

0.15.284.2.6 LRAEntryType `template<class Settings >`

`typedef Settings::EntryType smtrat::LRAModule< Settings >::LRAEntryType`

Type of an entry within the tableau.

0.15.284.2.7 LRATableau `template<class Settings >`

`typedef lra::Tableau<typename Settings::Tableau_settings, LRABoundType, LRAEntryType> smtrat::LRAModule< Settings >::LRATableau`

The tableau which is the main data structure maintained by the LRAModule responsible for the pivoting steps.

0.15.284.2.8 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.

0.15.284.2.9 LRAVariable `template<class Settings >`

`typedef lra::Variable<LRABoundType, LRAEntryType> smtrat::LRAModule< Settings >::LRAVariable`

A variable of the LRAModule, being either a original variable or a slack variable representing a linear polynomial.

0.15.284.2.10 SettingsType `template<class Settings >`

`typedef Settings smtrat::LRAModule< Settings >::SettingsType`

0.15.284.2.11 VarVariableMap `template<class Settings >`

`typedef carl::FastMap<carl::Variable, LRAVariable*> smtrat::LRAModule< Settings >::VarVariableMap`

Maps an original variable to it's corresponding LRAModule variable.

0.15.284.3 Member Enumeration Documentation

0.15.284.3.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

0.15.284.4 Constructor & Destructor Documentation

0.15.284.4.1 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.

0.15.284.4.2 ~LRAModule() `template<class Settings >`

`virtual smtrat::LRAModule< Settings >::~LRAModule ( ) [virtual]`

Destructs this LRAModule.

0.15.284.5 Member Function Documentation

0.15.284.5.1 add() `bool smtrat::Module::add (`

```
            ModuleInput::const_iterator _subformula ) [inherited]
```

The module has to take the given sub-formula of the received formula into account.

Parameters

_subformula	The sub-formula to take additionally into account.

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

0.15.284.5.2 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 >.

0.15.284.5.3 addCore() [1/2]

```
virtual bool smtrat::Module::addCore (
    [[maybe_unused] ] 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.

0.15.284.5.4 addCore() [2/2]

```
template<class Settings >
bool smtrat::LRAModule< Settings >::addCore (
    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.

0.15.284.5.5 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula
(
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.284.5.6 addLemma()

```
void smtrat::Module::addLemma (
    const FormulaT & _lemma,
    const LemmaType & _lt = LemmaType::NORMAL,
    const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline], [inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.284.5.7 addOrigin()

```
void smtrat::Module::addOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.284.5.8 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::addReceivedSubformulaToPassedFormula (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.284.5.9 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.284.5.10 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula,
            const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.284.5.11 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.

0.15.284.5.12 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns

All satisfying assignments, if existent.

0.15.284.5.13 anAnswerFound()

`bool smtrat::Module::anAnswerFound ( ) const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.284.5.14 answerFound()

`const smtrat::Conditionals& smtrat::Module::answerFound ( ) const [inline], [inherited]`

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.284.5.15 backendsModel()

`const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.284.5.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.284.5.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.284.5.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.284.5.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
```

```
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false )  [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.284.5.20 check()

```
Answer smtrat::Module::check (
bool _final = false,
bool _full = true,
carl::Variable _objective = carl::Variable::NO_VARIABLE )  [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.284.5.21 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.

0.15.284.5.22 checkCore_old() template<class Settings >

Answer smtrat::LRAModule< Settings >::checkCore_old ( )

0.15.284.5.23 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.

0.15.284.5.24 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.

0.15.284.5.25 checkNotEqualConstraints() template<class Settings >

Answer smtrat::LRAModule< Settings >::checkNotEqualConstraints (Answer *_result*)

0.15.284.5.26 cleanModel()

void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.284.5.27 clearLemmas()

void smtrat::Module::clearLemmas ( ) [inline], [inherited]

Deletes all yet found lemmas.

0.15.284.5.28 clearModel()

void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]

Clears the assignment, if any was found.

0.15.284.5.29 clearModels()

void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]

Clears all assignments, if any was found.

0.15.284.5.30 clearPassedFormula()

void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]

0.15.284.5.31 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

0.15.284.5.32 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.

0.15.284.5.33 collectTheoryPropagations()

```
void smtrat::Module::collectTheoryPropagations ( ) [inherited]
```

0.15.284.5.34 constraintsToInform()

```
const carl::FastSet<FormulaT>& smtrat::Module::constraints↩
ToInform ( ) const [inline], [inherited]
```

Returns

The constraints which the backends must be informed about.

0.15.284.5.35 createInfeasibleSubsets()

```
template<class Settings >
void smtrat::LRAModule< Settings >::createInfeasibleSubsets (
    lra::EntryID _tableauEntry )
```

0.15.284.5.36 currentlySatisfied()

```
template<class Settings >
unsigned smtrat::LRAModule< Settings >::currentlySatisfied (
    const FormulaT & ) const [virtual]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise

Reimplemented from smtrat::Module.

0.15.284.5.37 currentlySatisfiedByBackend()

```
unsigned smtrat::Module::currentlySatisfiedByBackend (
    const FormulaT & _formula ) const [inherited]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.284.5.38 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.

0.15.284.5.39 deinformCore() [1/2]

```
virtual void smtrat::Module::deinformCore (
    [[maybe_unused] ] 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.

0.15.284.5.40 deinformCore() [2/2]

```
template<class Settings >
void smtrat::LRAModule< Settings >::deinformCore (
    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.

0.15.284.5.41 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

0.15.284.5.42 eraseSubformulaFromPassedFormula()

```
ModuleInput::iterator smtrat::Module::erase↩
SubformulaFromPassedFormula (
    ModuleInput::iterator _subformula,
    bool _ignoreOrigins = false ) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_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 >.

0.15.284.5.43 excludeNotReceivedVariablesFromModel()

`void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

0.15.284.5.44 findBestOrigin()

`std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin ( const std::vector< FormulaT > & _origins ) const [protected], [inherited]`

Parameters

_origins	

Returns

0.15.284.5.45 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.

0.15.284.5.46 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.

0.15.284.5.47 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.284.5.48 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.284.5.49 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.284.5.50 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.284.5.51 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.284.5.52 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.

0.15.284.5.53 getInformationRelevantFormulas() `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.284.5.54 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.

0.15.284.5.55 getOrigins() [1/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.284.5.56 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.284.5.57 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.

0.15.284.5.58 getRationalModel()

```
template<class Settings >
const RationalAssignment& 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.

0.15.284.5.59 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.

0.15.284.5.60 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.

0.15.284.5.61 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.

0.15.284.5.62 hasLemmas() `bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

0.15.284.5.63 hasValidInfeasibleSubset() `bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]`

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.284.5.64 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.284.5.65 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.

0.15.284.5.66 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.

0.15.284.5.67 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.

0.15.284.5.68 informCore() [1/2]

`virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.284.5.69 informCore() [2/2]

`template<class Settings > bool smtrat::LRAModule< Settings >::informCore ( const FormulaT & _constraint )`

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.

0.15.284.5.70 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.

0.15.284.5.71 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.

0.15.284.5.72 is_minimizing()

`bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.284.5.73 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.

0.15.284.5.74 isPreprocessor()

`bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.284.5.75 lemmas()

`const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.284.5.76 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.

0.15.284.5.77 model()

`const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.284.5.78 modelsDisjoint()

`bool smtrat::Module::modelsDisjoint ( const Model & _modelA, const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

Parameters

_modelA	The first model to check for.
_modelB	The second model to check for.

Returns

true, if there is no variable assigned by both models; false, otherwise.

0.15.284.5.79 moduleName()

`template<class Settings >`
`std::string smtrat::LRAModule< Settings >::moduleName ( ) const [inline], [virtual]`

Returns

The name of the given module type as name.

Reimplemented from smtrat::Module.

0.15.284.5.80 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

0.15.284.5.81 optimize()

`template<class Settings >`
`Answer smtrat::LRAModule< Settings >::optimize ( Answer _result )`

0.15.284.5.82 originalVariables()

`template<class Settings >`
`const VarVariableMap& smtrat::LRAModule< Settings >::originalVariables ( ) const [inline]`

Returns

A constant reference to the original variables.

0.15.284.5.83 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.284.5.84 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.284.5.85 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.284.5.86 pPassedFormula() `const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.284.5.87 pReceivedFormula() `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.284.5.88 print() `void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.284.5.89 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.

0.15.284.5.90 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.

0.15.284.5.91 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.

0.15.284.5.92 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.284.5.93 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.

0.15.284.5.94 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.

0.15.284.5.95 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.

0.15.284.5.96 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
```

```
const std::string & _initiation = "***" ) const  [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.284.5.97 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.

0.15.284.5.98 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
            const std::string & _initiation = "***" ) const  [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.284.5.99 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.

0.15.284.5.100 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.

0.15.284.5.101 probablyLooping()

```
bool smtrat::Module::probablyLooping (
    const typename Poly::PolyType & _branchingPolynomial,
    const Rational & _branchingValue ) const [protected], [inherited]
```

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.284.5.102 processLearnedBounds()

```
template<class Settings >
void smtrat::LRAModule< Settings >::processLearnedBounds ( )
```

0.15.284.5.103 processResult()

```
template<class Settings >
Answer smtrat::LRAModule< Settings >::processResult (
    Answer _result )
```

0.15.284.5.104 receivedFormulaChecked()

```
void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]
```

Notifies that the received formulas has been checked.

0.15.284.5.105 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩InfeasibleSubset (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

0.15.284.5.106 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const [inline], [inherited]
```

0.15.284.5.107 remove()

```
void smtrat::Module::remove (
    ModuleInput::const_iterator _subformula ) [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.
However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

Parameters

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

0.15.284.5.108 removeCore() [1/2]

`virtual void smtrat::Module::removeCore ( [[maybe_unused] ] 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.

0.15.284.5.109 removeCore() [2/2]

`template<class Settings > void smtrat::LRAModule< Settings >::removeCore ( ModuleInput::const_iterator _subformula )`

Removes everything related to the given sub-formula of the received formula.

Parameters

_subformula	The sub formula of the received formula to remove.

0.15.284.5.110 removeOrigin()

`std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin ( ModuleInput::iterator _formula, const FormulaT & _origin ) [inline], [protected], [inherited]`

0.15.284.5.111 removeOrigins()

`std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins ( ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]`

0.15.284.5.112 rPassedFormula()

`const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]`

Returns

A reference to the formula passed to the backends of this module.

0.15.284.5.113 rReceivedFormula()

`const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]`

Returns

A reference to the formula passed to this module.

0.15.284.5.114 runBackends() [1/2]

`virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]`

Reimplemented in smtrat::PModule.

0.15.284.5.115 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.

0.15.284.5.116 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.

0.15.284.5.117 setThreadPriority()

```
void smtrat::Module::setThreadPriority (
    thread_priority _threadPriority ) [inline], [inherited]
```

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.284.5.118 slackVariables()

```
template<class Settings >
const ExVariableMap& smtrat::LRAModule< Settings >::slackVariables ( ) const [inline]
```

Returns

A constant reference to the slack variables.

0.15.284.5.119 solverState()

```
Answer smtrat::Module::solverState ( ) const [inline], [inherited]
```

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.284.5.120 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 !=.

0.15.284.5.121 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.284.5.122 updateAllModels()

`void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.284.5.123 updateLemmas()

`void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.284.5.124 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.

0.15.284.5.125 usedBackends()

`const std::vector<Module*>& smtrat::Module::usedBackends ( ) const [inline], [inherited]`

Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.284.6 Field Documentation

0.15.284.6.1 mAllBackends

`std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.284.6.2 mAllModels

`std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.284.6.3 mBackendsFoundAnswer `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.284.6.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.284.6.5 mFinalCheck `bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.284.6.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.284.6.7 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.

0.15.284.6.8 mFirstUncheckedReceivedSubformula `ModuleInput::const_iterator smtrat::Module↩ ::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.284.6.9 mFoundAnswer `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.284.6.10 mFullCheck `bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.284.6.11 mInfeasibleSubsets `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.284.6.12 mInformedConstraints `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.284.6.13 mLastBranches `std::vector< Branching > smtrat::Module::mLastBranches = std↩ ::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.284.6.14 mLemmas `std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.284.6.15 mModel `Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.284.6.16 mModelComputed `bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.284.6.17 mNumOfBranchVarsToStore `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.284.6.18 mObjectiveVariable `carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.284.6.19 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

0.15.284.6.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.284.6.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.284.6.22 mSolverState `std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.284.6.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

0.15.284.6.24 mUsedBackends `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.284.6.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.285 smtrat::LVEModule< Settings > Class Template Reference

`#include <LVEModule.h>`

Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- LVEModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=nullptr)
- ~LVEModule ()
- void updateModel () const

  Updates the current assignment into the model.
- Answer checkCore ()

  Checks the received formula for consistency.
- bool isPreprocessor () const
- bool appliedPreprocessing () const
- bool add (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- void remove (ModuleInput::const_iterator _subformula)

  Removes everything related to the given sub-formula of the received formula.
- Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)

  Checks the received formula for consistency.
- Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- std::pair< bool, FormulaT > getReceivedFormulaSimplified ()
- bool inform (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void deinform (const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual void init ()

  Informs all backends about the so far encountered constraints, which have not yet been communicated.
- virtual void updateAllModels ()

  Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
- virtual std::list< std::vector< carl::Variable > > getModelEqualities () const

  Partition the variables from the current model into equivalence classes according to their assigned value.
- unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const
- virtual unsigned currentlySatisfied (const FormulaT &) const
- bool receivedVariable (carl::Variable::Arg _var) const
- Answer solverState () const
- std::size_t id () const
- void setId (std::size_t _id)

  Sets this modules unique ID to identify itself.
- thread_priority threadPriority () const
- void setThreadPriority (thread_priority _threadPriority)

  Sets the priority of this module to get a thread for running its check procedure.
- const ModuleInput * pReceivedFormula () const
- const ModuleInput & rReceivedFormula () const
- const ModuleInput * pPassedFormula () const
- const ModuleInput & rPassedFormula () const

- const Model & model () const
- const std::vector< Model > & allModels () const
- const std::vector< FormulaSetT > & infeasibleSubsets () const
- const std::vector< Module * > & usedBackends () const
- const carl::FastSet< FormulaT > & constraintsToInform () const
- const carl::FastSet< FormulaT > & informedConstraints () const
- void addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))

  Stores a lemma being a valid formula.
- bool hasLemmas ()

  Checks whether this module or any of its backends provides any lemmas.
- void clearLemmas ()

  Deletes all yet found lemmas.
- const std::vector< Lemma > & lemmas () const
- ModuleInput::const_iterator firstUncheckedReceivedSubformula () const
- ModuleInput::const_iterator firstSubformulaToPass () const
- void receivedFormulaChecked ()

  Notifies that the received formulas has been checked.
- const 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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.
- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)
- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.

- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

  Copies the infeasible subsets of the passed formula.
- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.285.1 Member Typedef Documentation

0.15.285.1.1 SettingsType

```
template<typename Settings >
typedef Settings smtrat::LVEModule< Settings >::SettingsType
```

0.15.285.2 Member Enumeration Documentation

0.15.285.2.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

0.15.285.3 Constructor & Destructor Documentation

0.15.285.3.1 LVEModule()

```
template<typename Settings >
smtrat::LVEModule< Settings >::LVEModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

0.15.285.3.2 ~LVEModule()

```
template<typename Settings >
smtrat::LVEModule< Settings >::~LVEModule ( )
```

0.15.285.4 Member Function Documentation

0.15.285.4.1 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.

0.15.285.4.2 addConstraintToInform()

```
void smtrat::Module::addConstraintToInform (
            const FormulaT & _constraint ) [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

0.15.285.4.3 addCore()

```
virtual bool smtrat::Module::addCore (
    [[maybe_unused] ] 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.

0.15.285.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula
(
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.285.4.5 addLemma()

```
void smtrat::Module::addLemma (
    const FormulaT & _lemma,
    const LemmaType & _lt = LemmaType::NORMAL,
    const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline],
[inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.285.4.6 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.

Parameters

_origin	A set of formulas in the received formula of this module.

0.15.285.4.7 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
            ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.285.4.8 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.285.4.9 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula,
            const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.285.4.10 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.

0.15.285.4.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns

All satisfying assignments, if existent.

0.15.285.4.12 anAnswerFound()

```
bool smtrat::Module::anAnswerFound ( ) const [inline], [protected],
[inherited]
```

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.285.4.13 answerFound()

```
const smtrat::Conditionals& smtrat::Module::answerFound ( ) const
[inline], [inherited]
```

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.285.4.14 appliedPreprocessing()

```
bool smtrat::PModule::appliedPreprocessing ( ) const [inline],
[inherited]
```

Returns

true, if the current received formula has been simplified and the result of this simplification is stored in the passed formula.

0.15.285.4.15 backendsModel()

```
const Model & smtrat::Module::backendsModel ( ) const [protected],
[inherited]
```

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.285.4.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.285.4.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.285.4.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.285.4.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.285.4.20 check()

```
Answer smtrat::PModule::check (
    bool _final = false,
    bool _full = true,
    carl::Variable _objective = carl::Variable::NO_VARIABLE ) [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.285.4.21 checkCore()

```
template<typename 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.

0.15.285.4.22 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.

0.15.285.4.23 checkModel()

```
unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]
```

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.285.4.24 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.285.4.25 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

0.15.285.4.26 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

0.15.285.4.27 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

0.15.285.4.28 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.285.4.29 collectOrigins() [1/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

0.15.285.4.30 collectOrigins() [2/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inherited]`

Collects the formulas in the given formula, which are part of the received formula.
If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.285.4.31 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

0.15.285.4.32 constraintsToInform() `const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns

The constraints which the backends must be informed about.

0.15.285.4.33 currentlySatisfied()

`virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.285.4.34 currentlySatisfiedByBackend()

`unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.285.4.35 deinform()

`void smtrat::Module::deinform ( const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.285.4.36 deinformCore()

`virtual void smtrat::Module::deinformCore ( [[maybe_unused] ] 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.

0.15.285.4.37 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

0.15.285.4.38 eraseSubformulaFromPassedFormula()

`ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula ( ModuleInput::iterator _subformula, bool _ignoreOrigins = false ) [protected], [virtual], [inherited]`

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_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 >.

0.15.285.4.39 excludeNotReceivedVariablesFromModel()

`void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

0.15.285.4.40 findBestOrigin()

`std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin ( const std::vector< FormulaT > & _origins ) const [protected], [inherited]`

Parameters

_origins	

Returns

0.15.285.4.41 firstSubformulaToPass()

`ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.285.4.42 firstUncheckedReceivedSubformula()

`ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.285.4.43 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.285.4.44 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasible↩Subset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.285.4.45 getBackendsAllModels()

`void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.285.4.46 getBackendsModel()

`void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.285.4.47 getInfeasibleSubsets() [1/2]

`void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.285.4.48 getInfeasibleSubsets() [2/2]

`std::vector< FormulaSetT > smtrat::Module::get↩InfeasibleSubsets ( 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.

0.15.285.4.49 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module↩::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.285.4.50 getModelEqualities()

`std::list< std::vector< carl::Variable > > smtrat::Module↩::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.285.4.51 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
const FormulaT & _formula,
FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.285.4.52 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
const FormulaT & _formula,
FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.285.4.53 getOrigins() [3/3]

```
const FormulaT& smtrat::Module::getOrigins (
ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]
```

Gets the origins of the passed formula at the given position.

Parameters

_formula	The position of a formula in the passed formulas.

Returns

The origins of the passed formula at the given position.

0.15.285.4.54 getReceivedFormulaSimplified()

```
std::pair< bool, FormulaT > smtrat::PModule::getReceivedFormulaSimplified ( ) [virtual], [inherited]
```

Returns

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's reveived formula, if the Boolean is true.

Reimplemented from smtrat::Module.

0.15.285.4.55 hasLemmas()

```
bool smtrat::Module::hasLemmas ( ) [inline], [inherited]
```

Checks whether this module or any of its backends provides any lemmas.

0.15.285.4.56 hasValidInfeasibleSubset()

```
bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]
```

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.285.4.57 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.285.4.58 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasible↩ Subsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.285.4.59 inform() `bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.
It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

Parameters

_constraint	The constraint to inform about.

Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

0.15.285.4.60 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.

0.15.285.4.61 informCore() `virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.285.4.62 informedConstraints()

`const carl::FastSet<FormulaT>& smtrat::Module::informedConstraints ( ) const [inline], [inherited]`

Returns

The position of the first constraint of which no backend has been informed about.

0.15.285.4.63 init()

`void smtrat::Module::init ( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not be called twice and only before the first runBackends call.
Reimplemented in smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntBlastModule< Settings >, smtrat::FPPModule< Settings >, smtrat::CurryModule< Settings >, and smtrat::BVModule< Settings >.

0.15.285.4.64 is_minimizing()

`bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.285.4.65 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.

0.15.285.4.66 isPreprocessor()

`bool smtrat::PModule::isPreprocessor ( ) const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.285.4.67 lemmas()

`const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.285.4.68 merge()

`std::vector< FormulaT > smtrat::Module::merge ( const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.
({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

Parameters

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns

The vector being the two given vectors merged.

0.15.285.4.69 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.285.4.70 modelsDisjoint() `bool smtrat::Module::modelsDisjoint ( const Model & _modelA, const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

Parameters

_modelA	The first model to check for.
_modelB	The second model to check for.

Returns

true, if there is no variable assigned by both models; false, otherwise.

0.15.285.4.71 moduleName() `template<typename Settings > std::string smtrat::LVEModule< Settings >::moduleName ( ) const [inline], [virtual]`

Returns

The name of the given module type as name.

Reimplemented from smtrat::Module.

0.15.285.4.72 objective() `carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

0.15.285.4.73 originInReceivedFormula() `bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.285.4.74 passedFormulaBegin() `ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.285.4.75 passedFormulaEnd() `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.285.4.76 pPassedFormula() `const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.285.4.77 pReceivedFormula() `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.285.4.78 print() `void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.285.4.79 printAllModels() `void smtrat::Module::printAllModels ( std::ostream & _out = std::cout ) [inherited]`

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.285.4.80 printInfeasibleSubsets() `void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.285.4.81 printModel() `void smtrat::Module::printModel ( std::ostream & _out = std::cout ) const [inherited]`

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.285.4.82 printPassedFormula()

`void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.285.4.83 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.285.4.84 probablyLooping()

`bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.285.4.85 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.285.4.86 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

0.15.285.4.87 receivedVariable()

`bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

0.15.285.4.88 remove()

`void smtrat::PModule::remove ( ModuleInput::const_iterator _subformula ) [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.
However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

Parameters

_subformula	The sub formula of the received formula to remove.

Reimplemented from smtrat::Module.

0.15.285.4.89 removeCore()

```
virtual void smtrat::Module::removeCore (
    [[maybe_unused] ] 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.

0.15.285.4.90 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

0.15.285.4.91 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

0.15.285.4.92 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.285.4.93 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to this module.

0.15.285.4.94 runBackends() [1/2]

```
virtual Answer smtrat::PModule::runBackends ( ) [inline],
[virtual], [inherited]
```

Reimplemented from smtrat::Module.

0.15.285.4.95 runBackends() [2/2]

```
Answer smtrat::PModule::runBackends (
    bool _final,
    bool _full,
    carl::Variable _objective ) [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

Parameters

_final	true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT.
_full	false, if this module should avoid too expensive procedures and rather return unknown instead.
_objective	if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good.

Returns

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.285.4.96 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.

0.15.285.4.97 setThreadPriority()

```
void smtrat::Module::setThreadPriority (
    thread_priority _threadPriority ) [inline], [inherited]
```

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.285.4.98 solverState()

```
Answer smtrat::Module::solverState ( ) const [inline], [inherited]
```

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.285.4.99 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 !=.

0.15.285.4.100 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.285.4.101 updateAllModels() `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.285.4.102 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.285.4.103 updateModel() `template<typename 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.

0.15.285.4.104 usedBackends() `const std::vector<Module*>& smtrat::Module::usedBackends ( ) const [inline], [inherited]`

Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.285.5 Field Documentation

0.15.285.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.285.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.285.5.3 mBackendsFoundAnswer `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.285.5.4 mConstraintsToInform

`carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.285.5.5 mFinalCheck

`bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.285.5.6 mFirstPosInLastBranches

`std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.285.5.7 mFirstSubformulaToPass

`ModuleInput::iterator smtrat::Module::mFirstSubformulaTo↩ Pass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.285.5.8 mFirstUncheckedReceivedSubformula

`ModuleInput::const_iterator smtrat::Module↩ ::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.285.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.285.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.285.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.285.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.285.5.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std↩ ::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.285.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.285.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.285.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.285.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.285.5.18 mObjectiveVariable

`carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.285.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

0.15.285.5.20 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.285.5.21 mSmallerMusesCheckCounter

`unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.285.5.22 mSolverState

`std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.285.5.23 mTheoryPropagations

`std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

0.15.285.5.24 mUsedBackends

`std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.285.5.25 mVariableCounters

`std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.286 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

0.15.286.1 Detailed Description

Base class for solvers.
This is the interface to the user.

0.15.286.2 Constructor & Destructor Documentation

0.15.286.2.1 Manager() `smtrat::Manager::Manager ( )`

Constructs a manager.

0.15.286.2.2 ~Manager() `smtrat::Manager::~Manager ( )`

Destructs a manager.

0.15.286.3 Member Function Documentation

0.15.286.3.1 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.

0.15.286.3.2 addBackend() [1/2]

```
template<typename Module >
BackendLink smtrat::Manager::addBackend (
            const BackendLink & backend ) [inline], [protected]
```

0.15.286.3.3 addBackend() [2/2]

```
template<typename Module >
BackendLink smtrat::Manager::addBackend (
            const std::initializer_list< BackendLink > & backends = {} ) [inline], [protected]
```

0.15.286.3.4 addEdge()

```
BackendLink& smtrat::Manager::addEdge (
            std::size_t from,
            std::size_t to ) [inline], [protected]
```

0.15.286.3.5 addInformationRelevantFormula()

```
void smtrat::Manager::addInformationRelevant↩
Formula (
            const FormulaT & formula ) [inline]
```

Adds formula to InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.286.3.6 allModels()

```
const std::vector<Model>& smtrat::Manager::allModels ( ) const [inline]
```

Returns

A list of all assignments, such that they satisfy the conjunction of the so far added formulas.

Note, that an assignment is only provided if the conjunction of so far added formulas is satisfiable. Furthermore, when solving non-linear real arithmetic formulas the assignment could contain other variables or freshly introduced variables.

0.15.286.3.7 check()

```
Answer smtrat::Manager::check (
            bool _full = true )
```

Checks the so far added formulas for satisfiability.

Returns

True, if the conjunction of the so far added formulas is satisfiable; False, if it not satisfiable; Unknown, if this solver cannot decide whether it is satisfiable or not.

0.15.286.3.8 clear() `void smtrat::Manager::clear ( ) [inline]`

0.15.286.3.9 clearInformationRelevantFormula() `void smtrat::Manager::clearInformationRelevant↩ Formula ( ) [inline]`

Deletes all InformationRelevantFormula.

0.15.286.3.10 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.

0.15.286.3.11 formula() `const ModuleInput& smtrat::Manager::formula ( ) const [inline]`

Returns

The conjunction of so far added formulas.

0.15.286.3.12 formulaBegin() `ModuleInput::iterator smtrat::Manager::formulaBegin ( ) [inline]`

0.15.286.3.13 formulaEnd() `ModuleInput::iterator smtrat::Manager::formulaEnd ( ) [inline]`

0.15.286.3.14 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.

0.15.286.3.15 getAllGeneratedModules() `const std::vector<Module*>& smtrat::Manager::getAll↩ GeneratedModules ( ) const [inline]`

Returns

All instantiated modules of the solver belonging to this manager.

0.15.286.3.16 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.

0.15.286.3.17 getInformationRelevantFormulas()

```
const std::set<FormulaT>& smtrat::Manager::getInformationRelevantFormulas ( ) [inline], [protected]
```

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.286.3.18 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.

0.15.286.3.19 getModelEqualities()

```
std::list< std::vector< carl::Variable > > smtrat::Manager::getModelEqualities ( ) const
```

Determines variables assigned by the currently found satisfying assignment to an equal value in their domain.

Returns

A list of vectors of variables, stating that the variables in one vector are assigned to equal values.

0.15.286.3.20 infeasibleSubsets()

```
const std::vector< FormulaSetT > & smtrat::Manager::infeasibleSubsets ( ) const
```

Returns

All infeasible subsets of the set so far added formulas.

Note, that the conjunction of the so far added formulas must be inconsistent to receive an infeasible subset.

0.15.286.3.21 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.

0.15.286.3.22 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.

0.15.286.3.23 lemmas() `std::vector< FormulaT > smtrat::Manager::lemmas ( )`

Returns the lemmas/tautologies which were made during the last solving provoked by check().
These lemmas can be used in the same manner as infeasible subsets are used.

Returns

The lemmas/tautologies made during solving.

0.15.286.3.24 logic() `auto smtrat::Manager::logic ( ) const [inline]`

Returns

A constant reference to the logic this solver considers.

0.15.286.3.25 model() `const Model & smtrat::Manager::model ( ) const`

Returns

An assignment of the variables, which occur in the so far added formulas, to values of their domains, such that it satisfies the conjunction of these formulas.

Note, that an assignment is only provided if the conjunction of so far added formulas is satisfiable. Furthermore, when solving non-linear real arithmetic formulas the assignment could contain other variables or freshly introduced variables.

0.15.286.3.26 objectiveVariable() `const carl::Variable& smtrat::Manager::objectiveVariable ( ) const [inline]`

0.15.286.3.27 pop() [1/2] `bool smtrat::Manager::pop ( void )`

Pops a backtrack point from the stack of backtrack points.
This provokes, that all formulas which have been added after that backtrack point are removed.
Note, that this interface has not necessarily be used to apply a solver constructed with SMT-RAT, but is often required by state-of-the-art SMT solvers when embedding a theory solver constructed with SMT-RAT into them.

0.15.286.3.28 pop() [2/2] `void smtrat::Manager::pop ( size_t _levels )`

0.15.286.3.29 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.

0.15.286.3.30 printAssignment() `void smtrat::Manager::printAssignment ( ) const`

Prints the currently found assignment of variables occurring in the so far added formulas to values of their domains, if the conjunction of these formulas is satisfiable.

0.15.286.3.31 printBackTrackStack() `void smtrat::Manager::printBackTrackStack ( std::ostream & _out = std::cout ) const`

Prints the stack of backtrack points.

Parameters

_out	The stream to print on.

0.15.286.3.32 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.

0.15.286.3.33 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.

0.15.286.3.34 push() `void smtrat::Manager::push ( void ) [inline]`

Pushes a backtrack point to the stack of backtrack points.
Note, that this interface has not necessarily be used to apply a solver constructed with SMT-RAT, but is often required by state-of-the-art SMT solvers when embedding a theory solver constructed with SMT-RAT into them.

0.15.286.3.35 rDebugOutputChannel() `std::ostream& smtrat::Manager::rDebugOutputChannel ( ) [inline]`

Returns

The stream to print the debug information on.

0.15.286.3.36 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.

0.15.286.3.37 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.

0.15.286.3.38 reset() `void smtrat::Manager::reset ( )`

0.15.286.3.39 rLogic() `auto& smtrat::Manager::rLogic ( ) [inline]`

Returns

A reference to the logic this solver considers.

0.15.286.3.40 setLemmaLevel() `void smtrat::Manager::setLemmaLevel ( LemmaLevel level ) [inline]`

Sets the current level for lemma generation.

Parameters

level	Level

0.15.286.3.41 setObjectiveVariable() `void smtrat::Manager::setObjectiveVariable ( carl::Variable var ) [inline]`

0.15.286.3.42 setStrategy() [1/2] `void smtrat::Manager::setStrategy ( const BackendLink & backend ) [inline], [protected]`

0.15.286.3.43 setStrategy() [2/2] `void smtrat::Manager::setStrategy ( const std::initializer_list< BackendLink > & backends ) [inline], [protected]`

0.15.286.4 Friends And Related Function Documentation

0.15.286.4.1 Module `friend class Module [friend]`

0.15.287 Minisat::Map< K, D, H, E > Class Template Reference

`#include <Map.h>`

Data Structures

- struct Pair

Public Member Functions

- Map ()
- Map (const H &h, const E &e)
- ~Map ()
- const D & operator[ ] (const K &k) const
- D & operator[ ] (const K &k)
- void insert (const K &k, const D &d)
- bool peek (const K &k, D &d) const
- bool has (const K &k) const
- void remove (const K &k)
- void clear ()
- int elems () const
- int bucket_count () const
- void moveTo (Map &other)
- const vec< Pair > & bucket (int i) const

0.15.287.1 Constructor & Destructor Documentation

0.15.287.1.1 Map() [1/2] `template<class K , class D , class H = Hash<K>, class E = Equal<K>> Minisat::Map< K, D, H, E >::Map ( ) [inline]`

0.15.287.1.2 Map() [2/2]

```
template<class K , class D , class H = Hash<K>, class E = Equal<K>>
Minisat::Map< K, D, H, E >::Map (
            const H & h,
            const E & e )  [inline]
```

0.15.287.1.3 ~Map()

```
template<class K , class D , class H = Hash<K>, class E = Equal<K>>
Minisat::Map< K, D, H, E >::~Map ( )  [inline]
```

0.15.287.2 Member Function Documentation

0.15.287.2.1 bucket()

```
template<class K , class D , class H = Hash<K>, class E = Equal<K>>
const vec<Pair>& Minisat::Map< K, D, H, E >::bucket (
            int i ) const  [inline]
```

0.15.287.2.2 bucket_count()

```
template<class K , class D , class H = Hash<K>, class E = Equal<↩
K>>
int Minisat::Map< K, D, H, E >::bucket_count ( ) const  [inline]
```

0.15.287.2.3 clear()

```
template<class K , class D , class H = Hash<K>, class E = Equal<K>>
void Minisat::Map< K, D, H, E >::clear ( )  [inline]
```

0.15.287.2.4 elems()

```
template<class K , class D , class H = Hash<K>, class E = Equal<K>>
int Minisat::Map< K, D, H, E >::elems ( ) const  [inline]
```

0.15.287.2.5 has()

```
template<class K , class D , class H = Hash<K>, class E = Equal<K>>
bool Minisat::Map< K, D, H, E >::has (
            const K & k ) const  [inline]
```

0.15.287.2.6 insert()

```
template<class K , class D , class H = Hash<K>, class E = Equal<K>>
void Minisat::Map< K, D, H, E >::insert (
            const K & k,
            const D & d )  [inline]
```

0.15.287.2.7 moveTo()

```
template<class K , class D , class H = Hash<K>, class E = Equal<K>>
void Minisat::Map< K, D, H, E >::moveTo (
            Map< K, D, H, E > & other )  [inline]
```

0.15.287.2.8 operator[]() [1/2]

```
template<class K , class D , class H = Hash<K>, class E = Equal<↩
K>>
D& Minisat::Map< K, D, H, E >::operator[] (
            const K & k )  [inline]
```

0.15.287.2.9 operator[]() [2/2]

```
template<class K , class D , class H = Hash<K>, class E = Equal<↩
K>>
const D& Minisat::Map< K, D, H, E >::operator[] (
            const K & k ) const  [inline]
```

0.15.287.2.10 peek()

```
template<class K , class D , class H = Hash<K>, class E = Equal<K>>
bool Minisat::Map< K, D, H, E >::peek (
            const K & k,
            D & d ) const  [inline]
```

0.15.287.2.11 remove()

```
template<class K , class D , class H = Hash<K>, class E = Equal<K>>
void Minisat::Map< K, D, H, E >::remove (
            const K & k )  [inline]
```

0.15.288 benchmax::MathSAT Class Reference

Tool wrapper for MathSAT for SMT-LIB.

`#include <MathSAT.h>`

Public Member Functions

- MathSAT (const fs::path &binary, const std::string &arguments)

  New MathSAT tool.
- virtual bool canHandle (const fs::path &path) const override

  Only handle .smt2 files.
- virtual void additionalResults (const fs::path &, BenchmarkResult &result) const override

  Computes the answer string from the exit code and parses statistics output.
- std::string name () const

  Common name of this tool.
- fs::path binary () const

  Full path to the binary.
- const std::map< std::string, std::string > & attributes () const

  A set of attributes, for example compilation options.
- std::vector< std::string > resolveDependencies () const

  Get dependencies of binary required to run it (via ldd)
- std::size_t attributeHash () const

  Hash of the attributes.
- virtual std::string getCommandline (const std::string &file) const

  Compose commandline for this tool and the given input file.
- virtual std::string getCommandline (const std::string &file, const std::string &localBinary) const

  Compose commandline for this tool with another binary name and the given input file.
- virtual std::optional< std::string > parseCommandline (const std::string &cmdline) const

  Compose commandline for this tool and the given input file.

Protected Attributes

- std::string mName

  (Non-unique) identifier for the tool.
- fs::path mBinary

  Path to the binary.
- std::string mArguments

  Command line arguments that should be passed to the binary.
- std::map< std::string, std::string > mAttributes

  Attributes of the tool obtained by introspection of the binary.

0.15.288.1 Detailed Description

Tool wrapper for MathSAT for SMT-LIB.

0.15.288.2 Constructor & Destructor Documentation

0.15.288.2.1 MathSAT()

```
benchmax::MathSAT::MathSAT (
    const fs::path & binary,
    const std::string & arguments ) [inline]
```

New MathSAT tool.

0.15.288.3 Member Function Documentation

0.15.288.3.1 additionalResults()

```
virtual void benchmax::MathSAT::additionalResults (
    const fs::path & ,
    BenchmarkResult & result ) const [inline], [override], [virtual]
```

Computes the answer string from the exit code and parses statistics output.
Reimplemented from benchmax::Tool.

0.15.288.3.2 attributeHash()

```
std::size_t benchmax::Tool::attributeHash ( ) const [inline], [inherited]
```

Hash of the attributes.

0.15.288.3.3 attributes()

```
const std::map<std::string,std::string>& benchmax::Tool::attributes ( ) const [inline], [inherited]
```

A set of attributes, for example compilation options.

0.15.288.3.4 binary()

```
fs::path benchmax::Tool::binary ( ) const [inline], [inherited]
```

Full path to the binary.

0.15.288.3.5 canHandle()

```
virtual bool benchmax::MathSAT::canHandle (
    const fs::path & path ) const [inline], [override], [virtual]
```

Only handle .smt2 files.
Reimplemented from benchmax::Tool.

0.15.288.3.6 getCommandline() [1/2]

```
virtual std::string benchmax::Tool::getCommandline (
    const std::string & file ) const [inline], [virtual], [inherited]
```

Compose commandline for this tool and the given input file.

0.15.288.3.7 getCommandline() [2/2]

```
virtual std::string benchmax::Tool::getCommandline (
    const std::string & file,
    const std::string & localBinary ) const [inline], [virtual], [inherited]
```

Compose commandline for this tool with another binary name and the given input file.

0.15.288.3.8 name() `std::string benchmax::Tool::name ( ) const [inline], [inherited]`

Common name of this tool.

0.15.288.3.9 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.

0.15.288.3.10 resolveDependencies() `std::vector<std::string> benchmax::Tool::resolveDependencies ( ) const [inline], [inherited]`

Get dependencies of binary required to run it (via ldd)

0.15.288.4 Field Documentation

0.15.288.4.1 mArguments `std::string benchmax::Tool::mArguments [protected], [inherited]`

Command line arguments that should be passed to the binary.

0.15.288.4.2 mAttributes `std::map<std::string,std::string> benchmax::Tool::mAttributes [protected], [inherited]`

Attributes of the tool obtained by introspection of the binary.

0.15.288.4.3 mBinary `fs::path benchmax::Tool::mBinary [protected], [inherited]`

Path to the binary.

0.15.288.4.4 mName `std::string benchmax::Tool::mName [protected], [inherited]`

(Non-unique) identifier for the tool.

0.15.289 smtrat::mcsat::fm::MaxSizeComparator Struct Reference

This heuristic chooses the explanation excluding the largest interval.

`#include <ConflictGenerator.h>`

Public Member Functions

- bool operator() (const Bound &b1, const Bound &b2) const

Data Fields

- bool symmetric = true

0.15.289.1 Detailed Description

This heuristic chooses the explanation excluding the largest interval.

0.15.289.2 Member Function Documentation

0.15.289.2.1 operator()() `bool smtrat::mcsat::fm::MaxSizeComparator::operator() ( const Bound & b1, const Bound & b2 ) const [inline]`

0.15.289.3 Field Documentation

0.15.289.3.1 symmetric `bool smtrat::mcsat::fm::MaxSizeComparator::symmetric = true`

0.15.290 smtrat::MaxSMT< Solver, Strategy > Class Template Reference

`#include <MaxSMT.h>`

Public Member Functions

- MaxSMT (Solver &s)
- void add_soft_formula (const FormulaT &formula, Rational weight, const std::string &id)
- void remove_soft_formula (const FormulaT &formula)
- void reset ()
- bool active () const
- std::tuple< Answer, Model, ObjectiveValues > compute ()

0.15.290.1 Constructor & Destructor Documentation

0.15.290.1.1 MaxSMT()

```
template<typename Solver , MaxSMTStrategy Strategy>
smtrat::MaxSMT< Solver, Strategy >::MaxSMT (
            Solver & s )  [inline]
```

0.15.290.2 Member Function Documentation

0.15.290.2.1 active()

```
template<typename Solver , MaxSMTStrategy Strategy>
bool smtrat::MaxSMT< Solver, Strategy >::active ( ) const  [inline]
```

0.15.290.2.2 add_soft_formula()

```
template<typename Solver , MaxSMTStrategy Strategy>
void smtrat::MaxSMT< Solver, Strategy >::add_soft_formula (
            const FormulaT & formula,
            Rational weight,
            const std::string & id )  [inline]
```

0.15.290.2.3 compute()

```
template<typename Solver , MaxSMTStrategy Strategy>
std::tuple<Answer,Model,ObjectiveValues> smtrat::MaxSMT< Solver, Strategy >::compute ( )
[inline]
```

0.15.290.2.4 remove_soft_formula()

```
template<typename Solver , MaxSMTStrategy Strategy>
void smtrat::MaxSMT< Solver, Strategy >::remove_soft_formula (
            const FormulaT & formula )  [inline]
```

0.15.290.2.5 reset()

```
template<typename Solver , MaxSMTStrategy Strategy>
void smtrat::MaxSMT< Solver, Strategy >::reset ( )  [inline]
```

0.15.291 smtrat::maxsmt::MaxSMTBackend< Solver, Strategy > Class Template Reference

#include <MaxSMT.h>

0.15.292 smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::FU_MALIK_INCREMENTAL > Class Template Reference

#include <MaxSMT_FuMalikIncremental.h>

Public Member Functions

- MaxSMTBackend (Solver &solver, const std::vector< FormulaT > &softClauses)
- Answer run ()

0.15.292.1 Constructor & Destructor Documentation

0.15.292.1.1 MaxSMTBackend() template<typename Solver >

```
smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::FU_MALIK_INCREMENTAL >::MaxSMTBackend (
            Solver & solver,
            const std::vector< FormulaT > & softClauses )  [inline]
```

0.15.292.2 Member Function Documentation

0.15.292.2.1 run() template<typename Solver >

```
Answer smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::FU_MALIK_INCREMENTAL >::run ( )
[inline]
```

0.15.293 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 ()

0.15.293.1 Constructor & Destructor Documentation

0.15.293.1.1 MaxSMTBackend() template<typename Solver >

```
smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::LINEAR_SEARCH >::MaxSMTBackend (
            Solver & solver,
            const std::vector< FormulaT > & softClauses )  [inline]
```

0.15.293.2 Member Function Documentation

0.15.293.2.1 run() template<typename Solver >

```
Answer smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::LINEAR_SEARCH >::run ( )  [inline]
```

0.15.294 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 ()

0.15.294.1 Constructor & Destructor Documentation

0.15.294.1.1 MaxSMTBackend() `template<typename Solver >`

```
smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::MSU3 >::MaxSMTBackend (
            Solver & solver,
            const std::vector< FormulaT > & softClauses )  [inline]
```

0.15.294.2 Member Function Documentation

0.15.294.2.1 run() `template<typename Solver >`

```
Answer smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::MSU3 >::run ( )  [inline]
```

0.15.295 smtrat::MCBModule< Settings > Class Template Reference

`#include <MCBModule.h>`

Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- MCBModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=nullptr)
- ~MCBModule ()
- Answer checkCore ()

  Checks the received formula for consistency.
- void updateModel () const

  Updates the model, if the solver has detected the consistency of the received formula, beforehand.
- bool isPreprocessor () const
- bool appliedPreprocessing () const
- bool add (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- void remove (ModuleInput::const_iterator _subformula)

  Removes everything related to the given sub-formula of the received formula.
- Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)

  Checks the received formula for consistency.
- Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- std::pair< bool, FormulaT > getReceivedFormulaSimplified ()
- bool inform (const FormulaT &_constraint)

Informs the module about the given constraint.

- void deinform (const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual void init ()

  Informs all backends about the so far encountered constraints, which have not yet been communicated.
- virtual void updateAllModels ()

  Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
- virtual std::list< std::vector< carl::Variable > > getModelEqualities () const

  Partition the variables from the current model into equivalence classes according to their assigned value.
- unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const
- virtual unsigned currentlySatisfied (const FormulaT &) const
- bool receivedVariable (carl::Variable::Arg _var) const
- Answer solverState () const
- std::size_t id () const
- void setId (std::size_t _id)

  Sets this modules unique ID to identify itself.
- thread_priority threadPriority () const
- void setThreadPriority (thread_priority _threadPriority)

  Sets the priority of this module to get a thread for running its check procedure.
- const ModuleInput * pReceivedFormula () const
- const ModuleInput & rReceivedFormula () const
- const ModuleInput * pPassedFormula () const
- const ModuleInput & rPassedFormula () const
- const Model & model () const
- const std::vector< Model > & allModels () const
- const std::vector< FormulaSetT > & infeasibleSubsets () const
- const std::vector< Module * > & usedBackends () const
- const carl::FastSet< FormulaT > & constraintsToInform () const
- const carl::FastSet< FormulaT > & informedConstraints () const
- void addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))

  Stores a lemma being a valid formula.
- bool hasLemmas ()

  Checks whether this module or any of its backends provides any lemmas.
- void clearLemmas ()

  Deletes all yet found lemmas.
- const std::vector< Lemma > & lemmas () const
- ModuleInput::const_iterator firstUncheckedReceivedSubformula () const
- ModuleInput::const_iterator firstSubformulaToPass () const
- void receivedFormulaChecked ()

  Notifies that the received formulas has been checked.
- const 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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] 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 _useReceivedFormulaAs↩ Premise=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 >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.295.1 Member Typedef Documentation

0.15.295.1.1 SettingsType

```
template<typename Settings >
typedef Settings smtrat::MCBModule< Settings >::SettingsType
```

0.15.295.2 Member Enumeration Documentation

0.15.295.2.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

0.15.295.3 Constructor & Destructor Documentation

0.15.295.3.1 MCBModule()

```
template<typename Settings >
smtrat::MCBModule< Settings >::MCBModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

0.15.295.3.2 ~MCBModule()

```
template<typename Settings >
smtrat::MCBModule< Settings >::~MCBModule ( )
```

0.15.295.4 Member Function Documentation

0.15.295.4.1 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.

0.15.295.4.2 addConstraintToInform() `void smtrat::Module::addConstraintToInform ( const FormulaT & _constraint ) [protected], [virtual], [inherited]`

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

0.15.295.4.3 addCore() `virtual bool smtrat::Module::addCore ( [[maybe_unused] ] 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.

0.15.295.4.4 addInformationRelevantFormula() `void smtrat::Module::addInformationRelevantFormula ( const FormulaT & formula ) [protected], [inherited]`

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.295.4.5 addLemma()

```
void smtrat::Module::addLemma (
    const FormulaT & _lemma,
    const LemmaType & _lt = LemmaType::NORMAL,
    const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline], [inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.295.4.6 addOrigin()

```
void smtrat::Module::addOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.295.4.7 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.295.4.8 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.295.4.9 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula,
            const FormulaT & _origin )  [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.295.4.10 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.

0.15.295.4.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const  [inline],
[inherited]
```

Returns

All satisfying assignments, if existent.

0.15.295.4.12 anAnswerFound()

```
bool smtrat::Module::anAnswerFound ( ) const  [inline], [protected],
[inherited]
```

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.295.4.13 answerFound() `const smtrat::Conditionals& smtrat::Module::answerFound ( ) const [inline], [inherited]`

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.295.4.14 appliedPreprocessing() `bool smtrat::PModule::appliedPreprocessing ( ) const [inline], [inherited]`

Returns

true, if the current received formula has been simplified and the result of this simplification is stored in the passed formula.

0.15.295.4.15 backendsModel() `const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.295.4.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.295.4.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.295.4.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.295.4.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
```

```
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false )  [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.295.4.20 check()

```
Answer smtrat::PModule::check (
bool _final = false,
bool _full = true,
carl::Variable _objective = carl::Variable::NO_VARIABLE )  [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.295.4.21 checkCore()

```
template<typename Settings >
Answer smtrat::MCBModule< Settings >::checkCore ( )  [virtual]
```

Checks the received formula for consistency.

Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.295.4.22 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.

0.15.295.4.23 checkModel()

```
unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]
```

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.295.4.24 cleanModel()

```
void smtrat::Module::cleanModel ( ) const [inline], [protected],
[inherited]
```

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.295.4.25 clearLemmas()

```
void smtrat::Module::clearLemmas ( ) [inline], [inherited]
```

Deletes all yet found lemmas.

0.15.295.4.26 clearModel()

```
void smtrat::Module::clearModel ( ) const [inline], [protected],
[inherited]
```

Clears the assignment, if any was found.

0.15.295.4.27 clearModels()

```
void smtrat::Module::clearModels ( ) const [inline], [protected],
[inherited]
```

Clears all assignments, if any was found.

0.15.295.4.28 clearPassedFormula()

```
void smtrat::Module::clearPassedFormula ( ) [protected],
[inherited]
```

0.15.295.4.29 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
const FormulaT & _formula,
FormulaSetT & _origins ) const [inherited]
```

0.15.295.4.30 collectOrigins() [2/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.
If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.295.4.31 collectTheoryPropagations()

```
void smtrat::Module::collectTheoryPropagations ( ) [inherited]
```

0.15.295.4.32 constraintsToInform()

```
const carl::FastSet<FormulaT>& smtrat::Module::constraints↩
ToInform ( ) const [inline], [inherited]
```

Returns

The constraints which the backends must be informed about.

0.15.295.4.33 currentlySatisfied()

```
virtual unsigned smtrat::Module::currentlySatisfied (
    const FormulaT & ) const [inline], [virtual], [inherited]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.295.4.34 currentlySatisfiedByBackend()

```
unsigned smtrat::Module::currentlySatisfiedByBackend (
    const FormulaT & _formula ) const [inherited]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.295.4.35 deinform()

```
void smtrat::Module::deinform (
    const FormulaT & _constraint ) [inherited]
```

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.295.4.36 deinformCore()

`virtual void smtrat::Module::deinformCore ( [[maybe_unused] ] 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.

0.15.295.4.37 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

0.15.295.4.38 eraseSubformulaFromPassedFormula()

`ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula ( ModuleInput::iterator _subformula, bool _ignoreOrigins = false ) [protected], [virtual], [inherited]`

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_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 >.

0.15.295.4.39 excludeNotReceivedVariablesFromModel()

`void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

0.15.295.4.40 findBestOrigin()

`std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin ( const std::vector< FormulaT > & _origins ) const [protected], [inherited]`

Parameters

_origins	

Returns

0.15.295.4.41 firstSubformulaToPass() `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.295.4.42 firstUncheckedReceivedSubformula() `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.295.4.43 freeSplittingVariable() `static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.295.4.44 generateTrivialInfeasibleSubset() `void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.295.4.45 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.295.4.46 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.295.4.47 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.295.4.48 getInfeasibleSubsets() [2/2] `std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets ( const Module & _backend ) const [protected], [inherited]`

Get the infeasible subsets the given backend provides.
Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

Parameters

_backend	The backend from which to obtain the infeasible subsets.

Returns

The infeasible subsets the given backend provides.

0.15.295.4.49 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.295.4.50 getModelEqualities()

`std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.295.4.51 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.295.4.52 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.295.4.53 getOrigins() [3/3]

```
const FormulaT& smtrat::Module::getOrigins (
    ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]
```

Gets the origins of the passed formula at the given position.

Parameters

_formula	The position of a formula in the passed formulas.

Returns

The origins of the passed formula at the given position.

0.15.295.4.54 getReceivedFormulaSimplified() `std::pair< bool, FormulaT > smtrat::PModule::getReceivedFormulaSimplified ( ) [virtual], [inherited]`

Returns

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's reveived formula, if the Boolean is true.

Reimplemented from smtrat::Module.

0.15.295.4.55 hasLemmas() `bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

0.15.295.4.56 hasValidInfeasibleSubset() `bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]`

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.295.4.57 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.295.4.58 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.295.4.59 inform() `bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.
It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

Parameters

_constraint	The constraint to inform about.

Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

0.15.295.4.60 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.

0.15.295.4.61 informCore()

`virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.295.4.62 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.

0.15.295.4.63 init()

`void smtrat::Module::init ( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not be called twice and only before the first runBackends call.
Reimplemented in smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntBlastModule< Settings >, smtrat::FPPModule< Settings >, smtrat::CurryModule< Settings >, and smtrat::BVModule< Settings >.

0.15.295.4.64 is_minimizing()

`bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.295.4.65 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.

0.15.295.4.66 isPreprocessor() `bool smtrat::PModule::isPreprocessor ( ) const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.295.4.67 lemmas() `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.295.4.68 merge() `std::vector< FormulaT > smtrat::Module::merge ( const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.
({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

Parameters

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns

The vector being the two given vectors merged.

0.15.295.4.69 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.295.4.70 modelsDisjoint() `bool smtrat::Module::modelsDisjoint ( const Model & _modelA, const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

Parameters

_modelA	The first model to check for.
_modelB	The second model to check for.

Returns

true, if there is no variable assigned by both models; false, otherwise.

0.15.295.4.71 moduleName()

`virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]`

Returns

The name of the given module type as name.

Reimplemented in smtrat::VSModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::STropModule< Settings >, smtrat::SplitSOSModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntEqModule< Settings >, smtrat::IntBlastModule< Settings >, smtrat::IncWidthModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::GBModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::CurryModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::BVModule< Settings >, and smtrat::BEModule< Settings >.

0.15.295.4.72 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

0.15.295.4.73 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.295.4.74 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.295.4.75 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.295.4.76 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.295.4.77 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.295.4.78 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.295.4.79 printAllModels()

`void smtrat::Module::printAllModels ( std::ostream & _out = std::cout ) [inherited]`

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.295.4.80 printInfeasibleSubsets()

`void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.295.4.81 printModel()

`void smtrat::Module::printModel ( std::ostream & _out = std::cout ) const [inherited]`

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.295.4.82 printPassedFormula()

`void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.295.4.83 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.295.4.84 probablyLooping()

`bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.295.4.85 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.295.4.86 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

0.15.295.4.87 receivedVariable()

`bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

0.15.295.4.88 remove()

`void smtrat::PModule::remove ( ModuleInput::const_iterator _subformula ) [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.
However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

Parameters

_subformula	The sub formula of the received formula to remove.

Reimplemented from smtrat::Module.

0.15.295.4.89 removeCore()

`virtual void smtrat::Module::removeCore ( [[maybe_unused] ] 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.

0.15.295.4.90 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin (
            ModuleInput::iterator _formula,
            const FormulaT & _origin ) [inline], [protected], [inherited]
```

0.15.295.4.91 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

0.15.295.4.92 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.295.4.93 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to this module.

0.15.295.4.94 runBackends() [1/2]

```
virtual Answer smtrat::PModule::runBackends ( ) [inline],
[virtual], [inherited]
```

Reimplemented from smtrat::Module.

0.15.295.4.95 runBackends() [2/2]

```
Answer smtrat::PModule::runBackends (
            bool _final,
            bool _full,
            carl::Variable _objective ) [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

Parameters

_final	true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT.
_full	false, if this module should avoid too expensive procedures and rather return unknown instead.
_objective	if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good.

Returns

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.295.4.96 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.

0.15.295.4.97 setThreadPriority()

`void smtrat::Module::setThreadPriority ( thread_priority _threadPriority ) [inline], [inherited]`

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.295.4.98 solverState()

`Answer smtrat::Module::solverState ( ) const [inline], [inherited]`

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.295.4.99 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 !=.

0.15.295.4.100 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.295.4.101 updateAllModels() `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.295.4.102 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.295.4.103 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.

0.15.295.4.104 usedBackends() `const std::vector<Module*>& smtrat::Module::usedBackends ( ) const [inline], [inherited]`

Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.295.5 Field Documentation

0.15.295.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.295.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.295.5.3 mBackendsFoundAnswer `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.295.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.295.5.5 mFinalCheck `bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.295.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.295.5.7 mFirstSubformulaToPass

`ModuleInput::iterator smtrat::Module::mFirstSubformulaTo↩Pass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.295.5.8 mFirstUncheckedReceivedSubformula

`ModuleInput::const_iterator smtrat::Module↩::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.295.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.295.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.295.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.295.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.295.5.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std↩::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.295.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.295.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.295.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.295.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.295.5.18 mObjectiveVariable `carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.295.5.19 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]`

Reusable splitting variables.

0.15.295.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.295.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.295.5.22 mSolverState `std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.295.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]`

0.15.295.5.24 mUsedBackends `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.295.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.296 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
- void initVariables (const carl::Variables &variables)
- const auto & variables () const
- const auto & assignedVariables () const
- AssignmentOrConflict findAssignment (carl::Variable var) const
- AssignmentOrConflict isInfeasible (carl::Variable var, const FormulaT &f)
- Explanation explain (carl::Variable var, const FormulasT &reason, bool force_use_core=false) const

- Explanation explain (carl::Variable var, const FormulaT &f, const FormulasT &reason)
- bool isActive (const FormulaT &f) const

0.15.296.1 Member Function Documentation

0.15.296.1.1 assignedVariables()

```
template<typename Settings >
const auto& smtrat::mcsat::MCSATBackend< Settings >::assignedVariables ( ) const  [inline]
```

0.15.296.1.2 explain() [1/2]

```
template<typename Settings >
Explanation smtrat::mcsat::MCSATBackend< Settings >::explain (
            carl::Variable var,
            const FormulasT & reason,
            bool force_use_core = false ) const  [inline]
```

0.15.296.1.3 explain() [2/2]

```
template<typename Settings >
Explanation smtrat::mcsat::MCSATBackend< Settings >::explain (
            carl::Variable var,
            const FormulaT & f,
            const FormulasT & reason )  [inline]
```

0.15.296.1.4 findAssignment()

```
template<typename Settings >
AssignmentOrConflict smtrat::mcsat::MCSATBackend< Settings >::findAssignment (
            carl::Variable var ) const  [inline]
```

0.15.296.1.5 getModel()

```
template<typename Settings >
const auto& smtrat::mcsat::MCSATBackend< Settings >::getModel ( ) const  [inline]
```

0.15.296.1.6 getTrail()

```
template<typename Settings >
const auto& smtrat::mcsat::MCSATBackend< Settings >::getTrail ( ) const  [inline]
```

0.15.296.1.7 initVariables()

```
template<typename Settings >
void smtrat::mcsat::MCSATBackend< Settings >::initVariables (
            const carl::Variables & variables )  [inline]
```

0.15.296.1.8 isActive()

```
template<typename Settings >
bool smtrat::mcsat::MCSATBackend< Settings >::isActive (
            const FormulaT & f ) const  [inline]
```

0.15.296.1.9 isInfeasible()

```
template<typename Settings >
AssignmentOrConflict smtrat::mcsat::MCSATBackend< Settings >::isInfeasible (
            carl::Variable var,
            const FormulaT & f )  [inline]
```

0.15.296.1.10 popAssignment()

```
template<typename Settings >
void smtrat::mcsat::MCSATBackend< Settings >::popAssignment (
            carl::Variable v ) [inline]
```

0.15.296.1.11 popConstraint()

```
template<typename Settings >
void smtrat::mcsat::MCSATBackend< Settings >::popConstraint (
            const FormulaT & f ) [inline]
```

0.15.296.1.12 pushAssignment()

```
template<typename Settings >
void smtrat::mcsat::MCSATBackend< Settings >::pushAssignment (
            carl::Variable v,
            const ModelValue & mv,
            const FormulaT & f ) [inline]
```

0.15.296.1.13 pushConstraint()

```
template<typename Settings >
void smtrat::mcsat::MCSATBackend< Settings >::pushConstraint (
            const FormulaT & f ) [inline]
```

0.15.296.1.14 variables()

```
template<typename Settings >
const auto& smtrat::mcsat::MCSATBackend< Settings >::variables ( ) const [inline]
```

0.15.297 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 is_consistent ()

  Checks if any inconsistency was detected.
- bool is_consistent (Minisat::Var v)
- std::optional< Explanation > explainInconsistency ()

  Checks if the trail is consistent after the assignment on the current level.
- Explanation explainTheoryPropagation (Minisat::Lit literal)
- template<typename Constraints >
  void initVariables (const Constraints &c)
- bool isTheoryVar (Minisat::Var v) const
- const carl::Variable & carlVar (Minisat::Var v) const
- Minisat::Var minisatVar (const carl::Variable &v) const
- std::vector< Minisat::Var > theoryVarAbstractions () const
- bool hasUnassignedDep (Minisat::Var v) const
- std::size_t theoryLevel (Minisat::Var var) const
- std::size_t computeTheoryLevel (Minisat::Var var) const
- std::size_t computeTheoryLevel (const FormulaT &f) const
- Minisat::Lit getDecisionLiteral (Minisat::Var var) const
- int 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)

0.15.297.1 Constructor & Destructor Documentation

0.15.297.1.1 MCSATMixin()

```
template<typename Settings >
template<typename BaseModule >
smtrat::mcsat::MCSATMixin< Settings >::MCSATMixin (
            BaseModule & baseModule )  [inline], [explicit]
```

0.15.297.2 Member Function Documentation

0.15.297.2.1 addBooleanVariable()

```
template<typename Settings >
std::size_t smtrat::mcsat::MCSATMixin< Settings >::addBooleanVariable (
            Minisat::Var variable )
```

Add a variable, return the level it was inserted on.

0.15.297.2.2 assignedAtTrailIndex()

```
template<typename Settings >
int smtrat::mcsat::MCSATMixin< Settings >::assignedAtTrailIndex (
            Minisat::Var variable ) const [inline]
```

0.15.297.2.3 backtrackTo()

```
template<typename Settings >
bool smtrat::mcsat::MCSATMixin< Settings >::backtrackTo (
            Minisat::Lit literal )
```

Backtracks to the theory decision represented by the given literal.

0.15.297.2.4 carlVar()

```
template<typename Settings >
const carl::Variable& smtrat::mcsat::MCSATMixin< Settings >::carlVar (
            Minisat::Var v ) const [inline]
```

0.15.297.2.5 computeTheoryLevel() [1/2]

```
template<typename Settings >
std::size_t smtrat::mcsat::MCSATMixin< Settings >::computeTheoryLevel (
            const FormulaT & f ) const [inline]
```

0.15.297.2.6 computeTheoryLevel() [2/2]

```
template<typename Settings >
std::size_t smtrat::mcsat::MCSATMixin< Settings >::computeTheoryLevel (
            Minisat::Var var ) const [inline]
```

0.15.297.2.7 current() [1/2]

```
template<typename Settings >
TheoryLevel& smtrat::mcsat::MCSATMixin< Settings >::current ( ) [inline]
```

0.15.297.2.8 current() [2/2]

```
template<typename Settings >
const TheoryLevel& smtrat::mcsat::MCSATMixin< Settings >::current ( ) const [inline]
```

Returns the current theory level.

0.15.297.2.9 decisionLevel()

```
template<typename Settings >
int smtrat::mcsat::MCSATMixin< Settings >::decisionLevel (
            Minisat::Var v ) const [inline]
```

0.15.297.2.10 doBooleanAssignment()

```
template<typename Settings >
void smtrat::mcsat::MCSATMixin< Settings >::doBooleanAssignment (
            Minisat::Lit lit ) [inline]
```

Add a new constraint.

0.15.297.2.11 evaluateLiteral()

```
template<typename Settings >
Minisat::lbool smtrat::mcsat::MCSATMixin< Settings >::evaluateLiteral (
            Minisat::Lit lit ) const
```

Evaluate a literal in the theory, set lastReason to last theory decision involved.

0.15.297.2.12 explainInconsistency()

```
template<typename Settings >
std::optional<Explanation> smtrat::mcsat::MCSATMixin< Settings >::explainInconsistency ( )
[inline]
```

Checks if the trail is consistent after the assignment on the current level.
The trail must be consistent on the previous level.
Returns std::nullopt if consistent and an explanation.

0.15.297.2.13 explainTheoryPropagation()

```
template<typename Settings >
Explanation smtrat::mcsat::MCSATMixin< Settings >::explainTheoryPropagation (
            Minisat::Lit literal ) [inline]
```

0.15.297.2.14 fullConsistencyCheck()

```
template<typename Settings >
bool smtrat::mcsat::MCSATMixin< Settings >::fullConsistencyCheck ( ) const [inline]
```

0.15.297.2.15 get()

```
template<typename Settings >
const TheoryLevel& smtrat::mcsat::MCSATMixin< Settings >::get (
            std::size_t level ) const [inline]
```

Returns the respective theory level.

0.15.297.2.16 getDecisionLiteral()

```
template<typename Settings >
Minisat::Lit smtrat::mcsat::MCSATMixin< Settings >::getDecisionLiteral (
            Minisat::Var var ) const [inline]
```

0.15.297.2.17 hasUnassignedDep()

```
template<typename Settings >
bool smtrat::mcsat::MCSATMixin< Settings >::hasUnassignedDep (
            Minisat::Var v ) const [inline]
```

0.15.297.2.18 initVariables()

```
template<typename Settings >
template<typename Constraints >
void smtrat::mcsat::MCSATMixin< Settings >::initVariables (
            const Constraints & c ) [inline]
```

0.15.297.2.19 is_consistent() [1/2]

```
template<typename Settings >
bool smtrat::mcsat::MCSATMixin< Settings >::is_consistent ( ) [inline]
```

Checks if any inconsistency was detected.

0.15.297.2.20 is_consistent() [2/2]

```
template<typename Settings >
bool smtrat::mcsat::MCSATMixin< Settings >::is_consistent (
            Minisat::Var v ) [inline]
```

0.15.297.2.21 isAssignedTheoryVariable()

```
template<typename Settings >
bool smtrat::mcsat::MCSATMixin< Settings >::isAssignedTheoryVariable (
            const carl::Variable & var ) const [inline]
```

0.15.297.2.22 isBooleanDecisionFeasible()

```
template<typename Settings >
std::pair<bool, std::optional<Explanation> > smtrat::mcsat::MCSATMixin< Settings >::is↩
BooleanDecisionFeasible (
            Minisat::Lit lit,
            bool always_explain = false )
```

0.15.297.2.23 isFeasible()

```
template<typename Settings >
std::optional<Explanation> smtrat::mcsat::MCSATMixin< Settings >::isFeasible (
            const carl::Variable & var )  [inline]
```

0.15.297.2.24 isFormulaUnivariate()

```
template<typename Settings >
bool smtrat::mcsat::MCSATMixin< Settings >::isFormulaUnivariate (
            const FormulaT & formula,
            std::size_t level ) const
```

Checks whether the given formula is univariate on the given level.

0.15.297.2.25 isTheoryVar()

```
template<typename Settings >
bool smtrat::mcsat::MCSATMixin< Settings >::isTheoryVar (
            Minisat::Var v ) const  [inline]
```

0.15.297.2.26 level()

```
template<typename Settings >
std::size_t smtrat::mcsat::MCSATMixin< Settings >::level ( ) const  [inline]
```

0.15.297.2.27 makeTheoryDecision()

```
template<typename Settings >
std::variant<Explanation,FormulasT> smtrat::mcsat::MCSATMixin< Settings >::makeTheoryDecision
(
            const carl::Variable & var )  [inline]
```

0.15.297.2.28 maxDegree()

```
template<typename Settings >
std::size_t smtrat::mcsat::MCSATMixin< Settings >::maxDegree (
            const Minisat::Var & var )  [inline]
```

0.15.297.2.29 minisatVar()

```
template<typename Settings >
Minisat::Var smtrat::mcsat::MCSATMixin< Settings >::minisatVar (
            const carl::Variable & v ) const  [inline]
```

0.15.297.2.30 model()

```
template<typename Settings >
const Model& smtrat::mcsat::MCSATMixin< Settings >::model ( ) const  [inline]
```

0.15.297.2.31 printClause()

```
template<typename Settings >
void smtrat::mcsat::MCSATMixin< Settings >::printClause (
            std::ostream & os,
            Minisat::CRef clause ) const
```

Prints a single clause.

0.15.297.2.32 propagateBooleanDomain()

```
template<typename Settings >
std::pair<boost::tribool, std::optional<Explanation> > smtrat::mcsat::MCSATMixin< Settings
>::propagateBooleanDomain (
            Minisat::Lit lit )
```

0.15.297.2.33 pushTheoryDecision()

```
template<typename Settings >
void smtrat::mcsat::MCSATMixin< Settings >::pushTheoryDecision (
            const FormulaT & assignment,
            Minisat::Lit decisionLiteral )
```

Push a theory decision.

0.15.297.2.34 theoryAssignmentComplete()

```
template<typename Settings >
bool smtrat::mcsat::MCSATMixin< Settings >::theoryAssignmentComplete ( ) const  [inline]
```

0.15.297.2.35 theoryLevel()

```
template<typename Settings >
std::size_t smtrat::mcsat::MCSATMixin< Settings >::theoryLevel (
            Minisat::Var var ) const  [inline]
```

0.15.297.2.36 theoryVarAbstractions()

```
template<typename Settings >
std::vector<Minisat::Var> smtrat::mcsat::MCSATMixin< Settings >::theoryVarAbstractions ( )
const  [inline]
```

0.15.297.2.37 theoryVars()

```
template<typename Settings >
std::vector<Minisat::Var> smtrat::mcsat::MCSATMixin< Settings >::theoryVars (
            const Minisat::Var & var )  [inline]
```

0.15.297.2.38 topSemanticPropagation()

```
template<typename Settings >
Minisat::Var smtrat::mcsat::MCSATMixin< Settings >::topSemanticPropagation ( )  [inline]
```

Getter for semantic propagations.

0.15.297.2.39 undecidedBooleanVariables()

```
template<typename Settings >
const std::vector<Minisat::Var>& smtrat::mcsat::MCSATMixin< Settings >::undecidedBoolean↩
Variables ( ) const  [inline]
```

0.15.297.2.40 undoBooleanAssignment()

```
template<typename Settings >
void smtrat::mcsat::MCSATMixin< Settings >::undoBooleanAssignment (
            Minisat::Lit lit )  [inline]
```

Remove the last constraint.
f must be the same as the one passed to the last call of pushConstraint().

0.15.297.2.41 variable()

```
template<typename Settings >
carl::Variable smtrat::mcsat::MCSATMixin< Settings >::variable (
            std::size_t level ) const  [inline]
```

Retrieve already decided theory variables.

0.15.297.3 Friends And Related Function Documentation

0.15.297.3.1 operator<< template<typename Settings > template<typename Sett >

```
std::ostream& operator<< (
            std::ostream & os,
            const MCSATMixin< Sett > & mcm )  [friend]
```

0.15.298 smtrat::expression::simplifier::MergeSimplifier Struct Reference

```
#include <MergeSimplifier.h>
```

Public Member Functions

- const ExpressionContent * simplify (const NaryExpression &expr) const
- const ExpressionContent * operator() (const carl::Variable &expr) const
- const ExpressionContent * operator() (const ITEExpression &expr) const
- const ExpressionContent * operator() (const QuantifierExpression &expr) const
- const ExpressionContent * operator() (const UnaryExpression &expr) const
- const ExpressionContent * operator() (const BinaryExpression &expr) const
- const ExpressionContent * operator() (const NaryExpression &expr) const
- const ExpressionContent * operator() (const ExpressionContent *_ec) const

Protected Member Functions

- virtual const ExpressionContent * simplify (const carl::Variable &) const
- virtual const ExpressionContent * simplify (const ITEExpression &) const
- virtual const ExpressionContent * simplify (const QuantifierExpression &) const
- virtual const ExpressionContent * simplify (const UnaryExpression &) const
- virtual const ExpressionContent * simplify (const BinaryExpression &) const

0.15.298.1 Member Function Documentation

0.15.298.1.1 operator()() [1/7]

```
const ExpressionContent* smtrat::expression::simplifier::Base↩
Simplifier::operator() (
            const BinaryExpression & expr ) const  [inline], [inherited]
```

0.15.298.1.2 operator()() [2/7]

```
const ExpressionContent* smtrat::expression::simplifier::Base↩
Simplifier::operator() (
            const carl::Variable & expr ) const  [inline], [inherited]
```

0.15.298.1.3 operator()() [3/7]

```
const ExpressionContent* smtrat::expression::simplifier::Base↩
Simplifier::operator() (
            const ExpressionContent * _ec ) const  [inline], [inherited]
```

0.15.298.1.4 operator()() [4/7]

```
const ExpressionContent* smtrat::expression::simplifier::Base↩
Simplifier::operator() (
            const ITEExpression & expr ) const  [inline], [inherited]
```

0.15.298.1.5 operator()() [5/7]

```
const ExpressionContent* smtrat::expression::simplifier::Base↩
Simplifier::operator() (
            const NaryExpression & expr ) const  [inline], [inherited]
```

0.15.298.1.6 operator()() [6/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const QuantifierExpression & expr ) const  [inline], [inherited]
```

0.15.298.1.7 operator()() [7/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const UnaryExpression & expr ) const  [inline], [inherited]
```

0.15.298.1.8 simplify() [1/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const BinaryExpression &  ) const  [inline], [protected], [virtual], [inherited]
```

0.15.298.1.9 simplify() [2/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const carl::Variable &  ) const  [inline], [protected], [virtual], [inherited]
```

0.15.298.1.10 simplify() [3/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const ITEExpression &  ) const  [inline], [protected], [virtual], [inherited]
```

0.15.298.1.11 simplify() [4/6]

```
const ExpressionContent* smtrat::expression::simplifier::MergeSimplifier::simplify (
            const NaryExpression & expr ) const  [inline], [virtual]
```

Reimplemented from smtrat::expression::simplifier::BaseSimplifier.

0.15.298.1.12 simplify() [5/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const QuantifierExpression &  ) const  [inline], [protected], [virtual], [inherited]
```

0.15.298.1.13 simplify() [6/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const UnaryExpression &  ) const  [inline], [protected], [virtual], [inherited]
```

Reimplemented in smtrat::expression::simplifier::NegationSimplifier.

0.15.299 smtrat::covering_ng::formula::Minimal Class Reference

```
#include <FormulaEvaluation.h>
```

Public Types

- using FormulaComplexityOrdering = std::function< bool(const node_ds::Node &, const node_ds::Node &)>

Public Member Functions

- Minimal (FormulaComplexityOrdering formula_complexity_ordering)
- void set_formula (typename cadcells::Polynomial::ContextType c, const FormulaT &f)
- void extend_valuation (const cadcells::Assignment &ass)

  Updates the valuations.
- void revert_valuation (const cadcells::Assignment &ass)

- std::vector< boost::container::flat_set< cadcells::Constraint > > compute_implicants () const
- Valuation root_valuation () const

0.15.299.1 Member Typedef Documentation

0.15.299.1.1 FormulaComplexityOrdering

```
using smtrat::covering_ng::formula::Minimal::FormulaComplexityOrdering
= std::function<bool(const node_ds::Node&, const node_ds::Node&)>
```

0.15.299.2 Constructor & Destructor Documentation

0.15.299.2.1 Minimal()

```
smtrat::covering_ng::formula::Minimal::Minimal (
FormulaComplexityOrdering formula_complexity_ordering ) [inline]
```

0.15.299.3 Member Function Documentation

0.15.299.3.1 compute_implicants()

```
std::vector< boost::container::flat_set< cadcells::Constraint
> > smtrat::covering_ng::formula::Minimal::compute_implicants ( ) const
```

0.15.299.3.2 extend_valuation()

```
void smtrat::covering_ng::formula::Minimal::extend_valuation (
const cadcells::Assignment & ass )
```

Updates the valuations.
Assumes that ass is an extension of the previous assignment the formula has been evaluated with.

Parameters

f	
ass	

0.15.299.3.3 revert_valuation()

```
void smtrat::covering_ng::formula::Minimal::revert_valuation (
const cadcells::Assignment & ass )
```

0.15.299.3.4 root_valuation()

```
Valuation smtrat::covering_ng::formula::Minimal::root_valuation (
) const
```

0.15.299.3.5 set_formula()

```
void smtrat::covering_ng::formula::Minimal::set_formula (
typename cadcells::Polynomial::ContextType c,
const FormulaT & f )
```

0.15.300 benchmax::Minisat Class Reference

Tool wrapper for a Minisat solver.

```
#include <Minisat.h>
```

Public Member Functions

- Minisat (const fs::path &binary, const std::string &arguments)

Create tool.

- virtual bool canHandle (const fs::path &path) const override

  Only handles .opb files.
- virtual void additionalResults (const fs::path &, BenchmarkResult &result) const override

  Parse results from stdout.
- std::string name () const

  Common name of this tool.
- fs::path binary () const

  Full path to the binary.
- const std::map< std::string, std::string > & attributes () const

  A set of attributes, for example compilation options.
- std::vector< std::string > resolveDependencies () const

  Get dependencies of binary required to run it (via ldd)
- std::size_t attributeHash () const

  Hash of the attributes.
- virtual std::string getCommandline (const std::string &file) const

  Compose commandline for this tool and the given input file.
- virtual std::string getCommandline (const std::string &file, const std::string &localBinary) const

  Compose commandline for this tool with another binary name and the given input file.
- virtual std::optional< std::string > parseCommandline (const std::string &cmdline) const

  Compose commandline for this tool and the given input file.

Protected Attributes

- std::string mName

  (Non-unique) identifier for the tool.
- fs::path mBinary

  Path to the binary.
- std::string mArguments

  Command line arguments that should be passed to the binary.
- std::map< std::string, std::string > mAttributes

  Attributes of the tool obtained by introspection of the binary.

0.15.300.1 Detailed Description

Tool wrapper for a Minisat solver.

0.15.300.2 Constructor & Destructor Documentation

0.15.300.2.1 Minisat()

```
benchmax::Minisat::Minisat (
    const fs::path & binary,
    const std::string & arguments ) [inline]
```

Create tool.

0.15.300.3 Member Function Documentation

0.15.300.3.1 additionalResults()

```
virtual void benchmax::Minisat::additionalResults (
    const fs::path & ,
    BenchmarkResult & result ) const [inline], [override], [virtual]
```

Parse results from stdout.
Reimplemented from benchmax::Tool.

0.15.300.3.2 attributeHash() `std::size_t benchmax::Tool::attributeHash ( ) const [inline], [inherited]`

Hash of the attributes.

0.15.300.3.3 attributes() `const std::map<std::string,std::string>& benchmax::Tool::attributes ( ) const [inline], [inherited]`

A set of attributes, for example compilation options.

0.15.300.3.4 binary() `fs::path benchmax::Tool::binary ( ) const [inline], [inherited]`

Full path to the binary.

0.15.300.3.5 canHandle() `virtual bool benchmax::Minisat::canHandle ( const fs::path & path ) const [inline], [override], [virtual]`

Only handles .opb files.

Reimplemented from benchmax::Tool.

0.15.300.3.6 getCommandline() [1/2] `virtual std::string benchmax::Tool::getCommandline ( const std::string & file ) const [inline], [virtual], [inherited]`

Compose commandline for this tool and the given input file.

0.15.300.3.7 getCommandline() [2/2] `virtual std::string benchmax::Tool::getCommandline ( const std::string & file, const std::string & localBinary ) const [inline], [virtual], [inherited]`

Compose commandline for this tool with another binary name and the given input file.

0.15.300.3.8 name() `std::string benchmax::Tool::name ( ) const [inline], [inherited]`

Common name of this tool.

0.15.300.3.9 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.

0.15.300.3.10 resolveDependencies() `std::vector<std::string> benchmax::Tool::resolveDependencies ( ) const [inline], [inherited]`

Get dependencies of binary required to run it (via ldd)

0.15.300.4 Field Documentation

0.15.300.4.1 mArguments `std::string benchmax::Tool::mArguments [protected], [inherited]`

Command line arguments that should be passed to the binary.

0.15.300.4.2 mAttributes `std::map<std::string,std::string> benchmax::Tool::mAttributes [protected], [inherited]`

Attributes of the tool obtained by introspection of the binary.

0.15.300.4.3 mBinary `fs::path benchmax::Tool::mBinary [protected], [inherited]`

Path to the binary.

0.15.300.4.4 mName `std::string benchmax::Tool::mName [protected], [inherited]`

(Non-unique) identifier for the tool.

0.15.301 benchmax::Minisatp Class Reference

Tool wrapper for the Minisatp solver for pseudo-Boolean problems.

`#include <Minisatp.h>`

Public Member Functions

- Minisatp (const fs::path &binary, const std::string &arguments)

  Create tool, add "-v0" to arguments.
- virtual bool canHandle (const fs::path &path) const override

  Only handles .opb files.
- virtual void additionalResults (const fs::path &, BenchmarkResult &result) const override

  Parse results from stdout.
- std::string name () const

  Common name of this tool.
- fs::path binary () const

  Full path to the binary.
- const std::map< std::string, std::string > & attributes () const

  A set of attributes, for example compilation options.
- std::vector< std::string > resolveDependencies () const

  Get dependencies of binary required to run it (via ldd)
- std::size_t attributeHash () const

  Hash of the attributes.
- virtual std::string getCommandline (const std::string &file) const

  Compose commandline for this tool and the given input file.
- virtual std::string getCommandline (const std::string &file, const std::string &localBinary) const

  Compose commandline for this tool with another binary name and the given input file.
- virtual std::optional< std::string > parseCommandline (const std::string &cmdline) const

  Compose commandline for this tool and the given input file.

Protected Attributes

- std::string mName

  (Non-unique) identifier for the tool.
- fs::path mBinary

  Path to the binary.
- std::string mArguments

  Command line arguments that should be passed to the binary.
- std::map< std::string, std::string > mAttributes

  Attributes of the tool obtained by introspection of the binary.

0.15.301.1 Detailed Description

Tool wrapper for the Minisatp solver for pseudo-Boolean problems.

0.15.301.2 Constructor & Destructor Documentation

0.15.301.2.1 Minisatp()

```
benchmax::Minisatp::Minisatp (
    const fs::path & binary,
    const std::string & arguments ) [inline]
```

Create tool, add "-v0" to arguments.

0.15.301.3 Member Function Documentation

0.15.301.3.1 additionalResults()

```
virtual void benchmax::Minisatp::additionalResults (
    const fs::path & ,
    BenchmarkResult & result ) const [inline], [override], [virtual]
```

Parse results from stdout.
Reimplemented from benchmax::Tool.

0.15.301.3.2 attributeHash()

```
std::size_t benchmax::Tool::attributeHash ( ) const [inline], [inherited]
```

Hash of the attributes.

0.15.301.3.3 attributes()

```
const std::map<std::string,std::string>& benchmax::Tool::attributes ( ) const [inline], [inherited]
```

A set of attributes, for example compilation options.

0.15.301.3.4 binary()

```
fs::path benchmax::Tool::binary ( ) const [inline], [inherited]
```

Full path to the binary.

0.15.301.3.5 canHandle()

```
virtual bool benchmax::Minisatp::canHandle (
    const fs::path & path ) const [inline], [override], [virtual]
```

Only handles .opb files.
Reimplemented from benchmax::Tool.

0.15.301.3.6 getCommandline() [1/2]

```
virtual std::string benchmax::Tool::getCommandline (
    const std::string & file ) const [inline], [virtual], [inherited]
```

Compose commandline for this tool and the given input file.

0.15.301.3.7 getCommandline() [2/2]

```
virtual std::string benchmax::Tool::getCommandline (
    const std::string & file,
    const std::string & localBinary ) const [inline], [virtual], [inherited]
```

Compose commandline for this tool with another binary name and the given input file.

0.15.301.3.8 name() `std::string benchmax::Tool::name ( ) const [inline], [inherited]`

Common name of this tool.

0.15.301.3.9 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.

0.15.301.3.10 resolveDependencies() `std::vector<std::string> benchmax::Tool::resolveDependencies ( ) const [inline], [inherited]`

Get dependencies of binary required to run it (via ldd)

0.15.301.4 Field Documentation

0.15.301.4.1 mArguments `std::string benchmax::Tool::mArguments [protected], [inherited]`

Command line arguments that should be passed to the binary.

0.15.301.4.2 mAttributes `std::map<std::string,std::string> benchmax::Tool::mAttributes [protected], [inherited]`

Attributes of the tool obtained by introspection of the binary.

0.15.301.4.3 mBinary `fs::path benchmax::Tool::mBinary [protected], [inherited]`

Path to the binary.

0.15.301.4.4 mName `std::string benchmax::Tool::mName [protected], [inherited]`

(Non-unique) identifier for the tool.

0.15.302 smtrat::mcsat::fm::MinSizeComparator Struct Reference

This heuristic chooses the explanation excluding the smallest interval.

`#include <ConflictGenerator.h>`

Public Member Functions

- bool operator() (const Bound &b1, const Bound &b2) const

Data Fields

- bool symmetric = true

0.15.302.1 Detailed Description

This heuristic chooses the explanation excluding the smallest interval.

0.15.302.2 Member Function Documentation

0.15.302.2.1 operator()() `bool smtrat::mcsat::fm::MinSizeComparator::operator() ( const Bound & b1, const Bound & b2 ) const [inline]`

0.15.302.3 Field Documentation

0.15.302.3.1 symmetric `bool smtrat::mcsat::fm::MinSizeComparator::symmetric = true`

0.15.303 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

0.15.303.1 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.

0.15.303.2 Member Function Documentation

0.15.303.2.1 operator()() `bool smtrat::mcsat::fm::MinVarCountComparator::operator() ( const Bound & b1, const Bound & b2 ) const [inline]`

0.15.303.3 Field Documentation

0.15.303.3.1 symmetric `bool smtrat::mcsat::fm::MinVarCountComparator::symmetric = false`

0.15.304 smtrat::cad::MISGeneration< heuristic > Class Template Reference

`#include <MISGeneration.h>`

Public Member Functions

- template<typename CAD >
  void operator() (const CAD &cad, std::vector< FormulaSetT > &mis)
- void operator() (const CAD &cad, std::vector< FormulaSetT > &mis)
- void operator() (const CAD &cad, std::vector< FormulaSetT > &mis)
- void operator() (const CAD &cad, std::vector< FormulaSetT > &mis)
- void operator() (const CAD &cad, std::vector< FormulaSetT > &mis)
- void operator() (const CAD &cad, std::vector< FormulaSetT > &mis)
- void operator() (const CAD &cad, std::vector< FormulaSetT > &mis)

0.15.304.1 Member Function Documentation

0.15.304.1.1 operator()() [1/7] template<MISHeuristic heuristic>

```
template<typename CAD >
void smtrat::cad::MISGeneration< heuristic >::operator() (
            const CAD & cad,
            std::vector< FormulaSetT > & mis )
```

0.15.304.1.2 operator()() [2/7] void smtrat::cad::MISGeneration< MISHeuristic::TRIVIAL >::operator() (

```
            const CAD & cad,
            std::vector< FormulaSetT > & mis )
```

0.15.304.1.3 operator()() [3/7] void smtrat::cad::MISGeneration< MISHeuristic::GREEDY >::operator() (

```
            const CAD & cad,
            std::vector< FormulaSetT > & mis )
```

0.15.304.1.4 operator()() [4/7] void smtrat::cad::MISGeneration< MISHeuristic::GREEDY_PRE >↩ ::operator() (

```
            const CAD & cad,
            std::vector< FormulaSetT > & mis )
```

0.15.304.1.5 operator()() [5/7] void smtrat::cad::MISGeneration< MISHeuristic::EXACT >::operator() (

```
            const CAD & cad,
            std::vector< FormulaSetT > & mis )
```

0.15.304.1.6 operator()() [6/7] void smtrat::cad::MISGeneration< MISHeuristic::HYBRID >::operator() (

```
            const CAD & cad,
            std::vector< FormulaSetT > & mis )
```

0.15.304.1.7 operator()() [7/7] void smtrat::cad::MISGeneration< MISHeuristic::GREEDY_WEIGHTED >::operator() (

```
            const CAD & cad,
            std::vector< FormulaSetT > & mis )
```

0.15.305 smtrat::MixedSignEncoder Class Reference

```
#include <MixedSignEncoder.h>
```

Public Member Functions

- MixedSignEncoder ()
- bool canEncode (const ConstraintT &constraint)
- Rational encodingSize (const ConstraintT &constraint)
- std::string name ()
- std::optional< FormulaT > encode (const ConstraintT &constraint)

  Encodes an arbitrary constraint.

Data Fields

- std::size_t problem_size

Protected Member Functions

- std::optional< FormulaT > doEncode (const ConstraintT &constraint)
- FormulaT generateVarChain (const std::set< carl::Variable > &vars, carl::FormulaType type)

0.15.305.1 Constructor & Destructor Documentation

0.15.305.1.1 MixedSignEncoder() `smtrat::MixedSignEncoder::MixedSignEncoder ( ) [inline]`

0.15.305.2 Member Function Documentation

0.15.305.2.1 canEncode() `bool smtrat::MixedSignEncoder::canEncode ( const ConstraintT & constraint ) [virtual]`

Implements smtrat::PseudoBoolEncoder.

0.15.305.2.2 doEncode() `std::optional< FormulaT > smtrat::MixedSignEncoder::doEncode ( const ConstraintT & constraint ) [protected], [virtual]`

Implements smtrat::PseudoBoolEncoder.

0.15.305.2.3 encode() `std::optional< FormulaT > smtrat::PseudoBoolEncoder::encode ( const ConstraintT & constraint ) [inherited]`

Encodes an arbitrary constraint.

Returns

encoded formula

0.15.305.2.4 encodingSize() `Rational smtrat::MixedSignEncoder::encodingSize ( const ConstraintT & constraint ) [virtual]`

Reimplemented from smtrat::PseudoBoolEncoder.

0.15.305.2.5 generateVarChain() `FormulaT smtrat::PseudoBoolEncoder::generateVarChain ( const std::set< carl::Variable > & vars, carl::FormulaType type ) [protected], [inherited]`

0.15.305.2.6 name() `std::string smtrat::MixedSignEncoder::name ( ) [inline], [virtual]`

Reimplemented from smtrat::PseudoBoolEncoder.

0.15.305.3 Field Documentation

0.15.305.3.1 problem_size `std::size_t smtrat::PseudoBoolEncoder::problem_size [inherited]`

0.15.306 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.

0.15.306.1 Member Typedef Documentation

0.15.306.1.1 Constraints

```
template<typename Settings >
using smtrat::cad::BaseProjection< Settings >::Constraints = CADConstraints<Settings::backtracking>
[protected], [inherited]
```

0.15.306.2 Member Function Documentation

0.15.306.2.1 addEqConstraint() [1/2]

```
template<typename Settings >
carl::Bitset smtrat::cad::BaseProjection< Settings >::addEqConstraint (
            const Poly & p,
            std::size_t cid,
            bool isBound )  [inline], [inherited]
```

Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.

0.15.306.2.2 addEqConstraint() [2/2]

```
template<typename Settings >
virtual carl::Bitset smtrat::cad::BaseProjection< Settings >::addEqConstraint (
            const UPoly & p,
            std::size_t cid,
            bool isBound )  [inline], [virtual], [inherited]
```

Adds the given polynomial of an equational constraint to the projection.
Reimplemented in smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >.

0.15.306.2.3 addPolynomial() [1/2]

```
template<typename Settings >
carl::Bitset smtrat::cad::BaseProjection< Settings >::addPolynomial (
            const Poly & p,
            std::size_t cid,
            bool isBound )  [inline], [inherited]
```

Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.

0.15.306.2.4 addPolynomial() [2/2]

```
template<typename Settings >
virtual carl::Bitset smtrat::cad::BaseProjection< Settings >::addPolynomial (
            const UPoly & p,
            std::size_t cid,
            bool isBound )  [pure virtual], [inherited]
```

Adds the given polynomial to the projection.

Implemented in smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >, smtrat::cad::Projection< In
smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Increme
smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, and smtrat::cad::Projection< Incrementality::SIMPLE

0.15.306.2.5 callRemoveCallback()

```
template<typename Settings >
void smtrat::cad::BaseProjection< Settings >::callRemoveCallback (
            std::size_t level,
            const SampleLiftedWith & slw ) const [inline], [protected], [inherited]
```

0.15.306.2.6 canBePurgedByBounds()

```
template<typename Settings >
bool smtrat::cad::BaseProjection< Settings >::canBePurgedByBounds (
            const UPoly & p ) const [inline], [protected], [inherited]
```

Checks whether a polynomial can safely be ignored due to the bounds.

0.15.306.2.7 dim()

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::dim ( ) const [inline], [inherited]
```

Returns the dimension of the projection.

0.15.306.2.8 empty() [1/2]

```
template<typename Settings >
virtual bool smtrat::cad::BaseProjection< Settings >::empty ( ) [inline], [virtual], [inherited]
```

0.15.306.2.9 empty() [2/2]

```
template<typename Settings >
virtual bool smtrat::cad::BaseProjection< Settings >::empty (
            std::size_t level ) const [pure virtual], [inherited]
```

Implemented in smtrat::qe::cad::Projection< Settings >, smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED
smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::ModelBasedProjection< Increme
smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, and smtrat::cad::Projection< Incrementality::FULL, Backtracking::HID

0.15.306.2.10 exportAsDot()

```
template<typename Settings >
virtual void smtrat::cad::BaseProjection< Settings >::exportAsDot (
            std::ostream & ) const [inline], [virtual], [inherited]
```

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, and smtrat::cad::Projection< Incrementality::FULL,

0.15.306.2.11 freeID()

```
template<typename Settings >
void smtrat::cad::BaseProjection< Settings >::freeID (
            std::size_t level,
            std::size_t id ) [inline], [protected], [inherited]
```

Frees a currently used polynomial id for the given level.

0.15.306.2.12 getID()

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::getID (
            std::size_t level ) [inline], [protected], [inherited]
```

Returns a fresh polynomial id for the given level.

0.15.306.2.13 getOrigin() template<typename Settings >

```
virtual Origin smtrat::cad::BaseProjection< Settings >::getOrigin (
            std::size_t level,
            std::size_t id ) const [inline], [virtual], [inherited]
```

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, BT, Settings >.

0.15.306.2.14 getPolyForLifting() template<typename Settings >

```
OptionalID smtrat::cad::BaseProjection< Settings >::getPolyForLifting (
            std::size_t level,
            SampleLiftedWith & slw ) [inline], [inherited]
```

Get a polynomial from this level suited for lifting.

0.15.306.2.15 getPolynomialById() template<typename Settings >

```
virtual const UPoly& smtrat::cad::BaseProjection< Settings >::getPolynomialById (
            std::size_t level,
            std::size_t id ) const [pure virtual], [inherited]
```

Retrieves a polynomial from its id.

Implemented in smtrat::qe::cad::Projection< Settings >, smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::ModelBasedProjection< Increme smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, and smtrat::cad::Projection< Incrementality::FULL, Backtracking::HID

0.15.306.2.16 hasPolynomialById() template<typename Settings >

```
virtual bool smtrat::cad::BaseProjection< Settings >::hasPolynomialById (
            std::size_t level,
            std::size_t id ) const [pure virtual], [inherited]
```

Implemented in smtrat::qe::cad::Projection< Settings >, smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::ModelBasedProjection< Increme smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, and smtrat::cad::Projection< Incrementality::FULL, Backtracking::HID

0.15.306.2.17 isPurged() template<typename Settings >

```
bool smtrat::cad::BaseProjection< Settings >::isPurged (
            std::size_t level,
            std::size_t id ) [inline], [protected], [inherited]
```

0.15.306.2.18 removePolynomial() [1/2] template<typename Settings >

```
void smtrat::cad::BaseProjection< Settings >::removePolynomial (
            const Poly & p,
            std::size_t cid,
            bool isBound ) [inline], [inherited]
```

Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.

0.15.306.2.19 removePolynomial() [2/2] template<typename Settings >

```
virtual void smtrat::cad::BaseProjection< Settings >::removePolynomial (
            const UPoly & p,
            std::size_t cid,
            bool isBound ) [pure virtual], [inherited]
```

Removes the given polynomial from the projection.

Implemented in smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::ModelBasedProje smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, S smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >, and smtrat::cad::Projection< Incrementality::NONE, Backtracking::

0.15.306.2.20 reset()

```
template<typename Settings >
void smtrat::cad::BaseProjection< Settings >::reset ( ) [inline], [inherited]
```

Resets all datastructures, use the given variables from now on.

0.15.306.2.21 setRemoveCallback()

```
template<typename Settings >
template<typename F >
void smtrat::cad::BaseProjection< Settings >::setRemoveCallback (
            F && f ) [inline], [inherited]
```

Sets a callback that is called whenever polynomials are removed.

0.15.306.2.22 size() [1/2]

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::size ( ) const [inline], [inherited]
```

0.15.306.2.23 size() [2/2]

```
template<typename Settings >
virtual std::size_t smtrat::cad::BaseProjection< Settings >::size (
            std::size_t level ) const [pure virtual], [inherited]
```

Implemented in smtrat::qe::cad::Projection< Settings >, smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::ModelBasedProjection< Increme smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, and smtrat::cad::Projection< Incrementality::FULL, Backtracking::HID

0.15.306.2.24 var()

```
template<typename Settings >
carl::Variable smtrat::cad::BaseProjection< Settings >::var (
            std::size_t level ) const [inline], [protected], [inherited]
```

Returns the variable that corresponds to the given level, that is the variable eliminated in this level.

0.15.306.2.25 vars()

```
template<typename Settings >
const auto& smtrat::cad::BaseProjection< Settings >::vars ( ) const [inline], [inherited]
```

Returns the variables used for projection.

0.15.306.3 Field Documentation

0.15.306.3.1 mConstraints

```
template<typename Settings >
const Constraints& smtrat::cad::BaseProjection< Settings >::mConstraints [protected], [inherited]
```

0.15.306.3.2 mInfo

```
template<typename Settings >
ProjectionInformation smtrat::cad::BaseProjection< Settings >::mInfo [protected], [inherited]
```

Additional info on projection, projection levels and projection polynomials.

0.15.306.3.3 mLiftingQueues

```
template<typename Settings >
std::vector<PolynomialLiftingQueue<BaseProjection> > smtrat::cad::BaseProjection< Settings
>::mLiftingQueues [protected], [inherited]
```

List of lifting queues that can be used for incremental projection.

0.15.306.3.4 mOperator

```
template<typename Settings >
ProjectionOperator smtrat::cad::BaseProjection< Settings >::mOperator [protected], [inherited]
```

The projection operator.

0.15.306.3.5 mRemoveCallback `template<typename Settings >` `std::function<void(std::size_t, const SampleLiftedWith&)> smtrat::cad::BaseProjection< Settings >::mRemoveCallback [protected], [inherited]`

Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.

0.15.307 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)

0.15.307.1 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.

0.15.307.2 Constructor & Destructor Documentation

0.15.307.2.1 ModelBasedProjection()

```
template<typename Settings >
smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >↩
::ModelBasedProjection (
            const Constraints & c,
            const Model & model )  [inline]
```

0.15.307.3 Member Function Documentation

0.15.307.3.1 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.

0.15.307.3.2 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 >.

0.15.307.3.3 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.

0.15.307.3.4 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 >.

0.15.307.3.5 callRemoveCallback()

```
template<typename Settings >
void smtrat::cad::BaseProjection< Settings >::callRemoveCallback (
            std::size_t level,
            const SampleLiftedWith & slw ) const  [inline], [protected], [inherited]
```

0.15.307.3.6 canBePurgedByBounds()

```
template<typename Settings >
bool smtrat::cad::BaseProjection< Settings >::canBePurgedByBounds (
            const UPoly & p ) const  [inline], [protected], [inherited]
```

Checks whether a polynomial can safely be ignored due to the bounds.

0.15.307.3.7 dim() [1/2] template<typename Settings >

```
std::size_t smtrat::cad::BaseProjection< Settings >::dim ( ) const [inline], [inherited]
```

Returns the dimension of the projection.

0.15.307.3.8 dim() [2/2] template<typename Settings >

```
std::size_t smtrat::cad::BaseProjection< Settings >::dim [inline]
```

Returns the dimension of the projection.

0.15.307.3.9 empty() [1/2] template<typename Settings >

```
virtual bool smtrat::cad::BaseProjection< Settings >::empty ( ) [inline], [virtual], [inherited]
```

0.15.307.3.10 empty() [2/2] template<typename Settings >

```
bool smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings
>::empty (
            std::size_t level ) const [inline], [override], [virtual]
```

Returns whether the number of polynomials in this level is zero.
Implements smtrat::cad::BaseProjection< Settings >.

0.15.307.3.11 exportAsDot() template<typename Settings >

```
virtual void smtrat::cad::BaseProjection< Settings >::exportAsDot (
            std::ostream & ) const [inline], [virtual], [inherited]
```

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, and smtrat::cad::Projection< Incrementality::FULL,

0.15.307.3.12 freeID() template<typename Settings >

```
void smtrat::cad::BaseProjection< Settings >::freeID (
            std::size_t level,
            std::size_t id ) [inline], [protected], [inherited]
```

Frees a currently used polynomial id for the given level.

0.15.307.3.13 getID() template<typename Settings >

```
std::size_t smtrat::cad::BaseProjection< Settings >::getID (
            std::size_t level ) [inline], [protected], [inherited]
```

Returns a fresh polynomial id for the given level.

0.15.307.3.14 getOrigin() template<typename Settings >

```
virtual Origin smtrat::cad::BaseProjection< Settings >::getOrigin (
            std::size_t level,
            std::size_t id ) const [inline], [virtual], [inherited]
```

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, BT, Settings >.

0.15.307.3.15 getPolyForLifting() template<typename Settings >

```
OptionalID smtrat::cad::BaseProjection< Settings >::getPolyForLifting (
            std::size_t level,
            SampleLiftedWith & slw ) [inline], [inherited]
```

Get a polynomial from this level suited for lifting.

0.15.307.3.16 getPolynomialById() template<typename Settings >

```
const UPoly& smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED,
Settings >::getPolynomialById (
            std::size_t level,
            std::size_t id ) const  [inline], [override], [virtual]
```

Get the polynomial from this level with the given id.
Implements smtrat::cad::BaseProjection< Settings >.

0.15.307.3.17 hasPolynomialById() template<typename Settings >

```
bool smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings
>::hasPolynomialById (
            std::size_t level,
            std::size_t id ) const  [inline], [override], [virtual]
```

Implements smtrat::cad::BaseProjection< Settings >.

0.15.307.3.18 isPurged() template<typename Settings >

```
bool smtrat::cad::BaseProjection< Settings >::isPurged (
            std::size_t level,
            std::size_t id )  [inline], [protected], [inherited]
```

0.15.307.3.19 projectNewPolynomial() template<typename Settings >

```
carl::Bitset smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED,
Settings >::projectNewPolynomial (
            const ConstraintSelection &  = carl::Bitset(true) )  [inline]
```

Returns false, as the projection is not incremental.

0.15.307.3.20 projectNextLevel() template<typename Settings >

```
void smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings
>::projectNextLevel (
            std::size_t level )  [inline]
```

Performs the projection model-based for one level, using only polynomials with closest roots.

0.15.307.3.21 removePolynomial() [1/2] template<typename Settings >

```
void smtrat::cad::BaseProjection< Settings >::removePolynomial (
            const Poly & p,
            std::size_t cid,
            bool isBound )  [inline], [inherited]
```

Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.

0.15.307.3.22 removePolynomial() [2/2] template<typename Settings >

```
void smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings
>::removePolynomial (
            const UPoly & p,
            std::size_t cid,
            bool  )  [inline], [override], [virtual]
```

Removed the given polynomial from the projection.
Asserts that this polynomial was the one added last and has the given constraint id as origin. Calls the callback function for every level with a mask designating the polynomials removed from this level.
Implements smtrat::cad::BaseProjection< Settings >.

0.15.307.3.23 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.

0.15.307.3.24 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.

0.15.307.3.25 size() [1/4]

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::size ( ) const [inline], [inherited]
```

0.15.307.3.26 size() [2/4]

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::size (
            void ) [inline]
```

0.15.307.3.27 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 >.

0.15.307.3.28 size() [4/4]

```
template<typename Settings >
virtual std::size_t smtrat::cad::BaseProjection< Settings >::size (
            void )
```

0.15.307.3.29 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.

0.15.307.3.30 vars()

```
template<typename Settings >
const auto& smtrat::cad::BaseProjection< Settings >::vars ( ) const [inline], [inherited]
```

Returns the variables used for projection.

0.15.307.4 Friends And Related Function Documentation

0.15.307.4.1 operator<<

```
template<typename Settings >
template<typename S >
std::ostream& operator<< (
            std::ostream & os,
            const ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, S > & p
) [friend]
```

0.15.307.5 Field Documentation

0.15.307.5.1 mConstraints `template<typename Settings >`

`const Constraints& smtrat::cad::BaseProjection< Settings >::mConstraints [protected], [inherited]`

0.15.307.5.2 mInfo `template<typename Settings >`

`ProjectionInformation smtrat::cad::BaseProjection< Settings >::mInfo [protected], [inherited]`

Additional info on projection, projection levels and projection polynomials.

0.15.307.5.3 mRemoveCallback `template<typename Settings >`

`std::function<void(std::size_t, const SampleLiftedWith&)> smtrat::cad::BaseProjection< Settings >::mRemoveCallback [protected], [inherited]`

Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.

0.15.308 smtrat::Module Class Reference

A base class for all kind of theory solving methods.

`#include <Module.h>`

Data Structures

- struct Lemma
- struct ModuleStatistics
- struct TheoryPropagation

Public Types

- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- Module (const ModuleInput *_formula, Conditionals &_foundAnswer, Manager *_manager=nullptr, std::string module_name="Module")

  Constructs a module.
- virtual ~Module ()

  Destructs a module.
- bool inform (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void deinform (const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual void init ()

  Informs all backends about the so far encountered constraints, which have not yet been communicated.
- bool add (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- virtual Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_↩VARIABLE)

  Checks the received formula for consistency.
- virtual void remove (ModuleInput::const_iterator _subformula)

  Removes everything related to the given sub-formula of the received formula.
- virtual void updateModel () const

Updates the model, if the solver has detected the consistency of the received formula, beforehand.

- virtual void updateAllModels ()

  Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
- virtual std::list< std::vector< carl::Variable > > getModelEqualities () const

  Partition the variables from the current model into equivalence classes according to their assigned value.
- unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const
- virtual unsigned currentlySatisfied (const FormulaT &) const
- bool receivedVariable (carl::Variable::Arg _var) const
- Answer solverState () const
- std::size_t id () const
- void setId (std::size_t _id)

  Sets this 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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] 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 ([[maybe_unused]] 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 _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.308.1 Detailed Description

A base class for all kind of theory solving methods.

0.15.308.2 Member Enumeration Documentation

0.15.308.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong]`

Enumerator

NORMAL	
PERMANENT	

0.15.308.3 Constructor & Destructor Documentation

0.15.308.3.1 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.

Parameters

_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.

0.15.308.3.2 ~Module() `smtrat::Module::~Module ( ) [virtual]`

Destructs a module.

0.15.308.4 Member Function Documentation

0.15.308.4.1 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.

0.15.308.4.2 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 >.

0.15.308.4.3 addCore() `virtual bool smtrat::Module::addCore ( [[maybe_unused] ] 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.

0.15.308.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula (
    const FormulaT & formula ) [protected]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.308.4.5 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.

0.15.308.4.6 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.

0.15.308.4.7 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.

0.15.308.4.8 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.

0.15.308.4.9 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.

0.15.308.4.10 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.

0.15.308.4.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const  [inline]
```

Returns

All satisfying assignments, if existent.

0.15.308.4.12 anAnswerFound() `bool smtrat::Module::anAnswerFound ( ) const [inline], [protected]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.308.4.13 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.

0.15.308.4.14 backendsModel() `const Model & smtrat::Module::backendsModel ( ) const [protected]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.308.4.15 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected]
```

0.15.308.4.16 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected]
```

0.15.308.4.17 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected]
```

0.15.308.4.18 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
```

```
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false )  [protected]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.308.4.19 check()

```
Answer smtrat::Module::check (
bool _final = false,
bool _full = true,
carl::Variable _objective = carl::Variable::NO_VARIABLE )  [virtual]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.308.4.20 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::VSModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::SymmetryModule< Settings >, smtrat::STropModule< Settings >, smtrat::SplitSOSModule< Settings >,

smtrat::SATModule< Settings >, smtrat::PFEModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Se
smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >,
smtrat::MCBModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASetti
smtrat::LRAModule< LRASettings1 >, smtrat::IntEqModule< Settings >, smtrat::IntBlastModule< Settings >,
smtrat::IncWidthModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >,
smtrat::ICPModule< ICPSettings1 >, smtrat::ICEModule< Settings >, smtrat::GBPPModule< Settings >,
smtrat::GBModule< Settings >, smtrat::FPPModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::ESModule< Settings
smtrat::EMModule< Settings >, smtrat::CurryModule< Settings >, smtrat::CubeLIAModule< Settings >,
smtrat::CSplitModule< Settings >, smtrat::CoveringNGModule< Settings >, smtrat::CNFerModule, smtrat::BVModule< Settings >,
and smtrat::BEModule< Settings >.

0.15.308.4.21 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.

0.15.308.4.22 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.

0.15.308.4.23 cleanModel()

```
void smtrat::Module::cleanModel ( ) const [inline], [protected]
```

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.308.4.24 clearLemmas()

```
void smtrat::Module::clearLemmas ( ) [inline]
```

Deletes all yet found lemmas.

0.15.308.4.25 clearModel()

```
void smtrat::Module::clearModel ( ) const [inline], [protected]
```

Clears the assignment, if any was found.

0.15.308.4.26 clearModels()

```
void smtrat::Module::clearModels ( ) const [inline], [protected]
```

Clears all assignments, if any was found.

0.15.308.4.27 clearPassedFormula()

```
void smtrat::Module::clearPassedFormula ( ) [protected]
```

0.15.308.4.28 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const
```

0.15.308.4.29 collectOrigins() [2/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const
```

Collects the formulas in the given formula, which are part of the received formula.
If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.308.4.30 collectTheoryPropagations()

```
void smtrat::Module::collectTheoryPropagations ( )
```

0.15.308.4.31 constraintsToInform()

```
const carl::FastSet<FormulaT>& smtrat::Module::constraints↩
ToInform ( ) const [inline]
```

Returns

The constraints which the backends must be informed about.

0.15.308.4.32 currentlySatisfied()

```
virtual unsigned smtrat::Module::currentlySatisfied (
    const FormulaT & ) const [inline], [virtual]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.308.4.33 currentlySatisfiedByBackend()

```
unsigned smtrat::Module::currentlySatisfiedByBackend
(
    const FormulaT & _formula ) const
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.308.4.34 deinform()

```
void smtrat::Module::deinform (
    const FormulaT & _constraint )
```

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.308.4.35 deinformCore()

`virtual void smtrat::Module::deinformCore ( [[maybe_unused] ] 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.

0.15.308.4.36 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

0.15.308.4.37 eraseSubformulaFromPassedFormula()

`ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula ( ModuleInput::iterator _subformula, bool _ignoreOrigins = false ) [protected], [virtual]`

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_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 >.

0.15.308.4.38 excludeNotReceivedVariablesFromModel()

`void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const`

Excludes all variables from the current model, which do not occur in the received formula.

0.15.308.4.39 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
    const std::vector< FormulaT > & _origins ) const [protected]
```

Parameters

_origins	

Returns

0.15.308.4.40 firstSubformulaToPass()

```
ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline]
```

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.308.4.41 firstUncheckedReceivedSubformula()

```
ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline]
```

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.308.4.42 freeSplittingVariable()

```
static void smtrat::Module::freeSplittingVariable (
    const FormulaT & _splittingVariable ) [inline], [static]
```

0.15.308.4.43 generateTrivialInfeasibleSubset()

```
void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected]
```

Stores the trivial infeasible subset being the set of received formulas.

0.15.308.4.44 getBackendsAllModels()

```
void smtrat::Module::getBackendsAllModels ( ) const [protected]
```

Stores all models of a backend in the list of all models of this module.

0.15.308.4.45 getBackendsModel()

```
void smtrat::Module::getBackendsModel ( ) const [protected]
```

0.15.308.4.46 getInfeasibleSubsets() [1/2]

```
void smtrat::Module::getInfeasibleSubsets ( ) [protected]
```

Copies the infeasible subsets of the passed formula.

0.15.308.4.47 getInfeasibleSubsets() [2/2]

```
std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets (
    const Module & _backend ) const [protected]
```

Get the infeasible subsets the given backend provides.
Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

Parameters

_backend	The backend from which to obtain the infeasible subsets.

Returns

The infeasible subsets the given backend provides.

0.15.308.4.48 getInformationRelevantFormulas() `const std::set< FormulaT > & smtrat::Module↩ ::getInformationRelevantFormulas ( ) [protected]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.308.4.49 getModelEqualities() `std::list< std::vector< carl::Variable > > smtrat::Module↩ ::getModelEqualities ( ) const [virtual]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.308.4.50 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
const FormulaT & _formula,
FormulaSetT & _origins ) const [inline], [protected]
```

Parameters

_formula	
_origins	

0.15.308.4.51 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
const FormulaT & _formula,
FormulasT & _origins ) const [inline], [protected]
```

Parameters

_formula	
_origins	

0.15.308.4.52 getOrigins() [3/3]

```
const FormulaT& smtrat::Module::getOrigins (
ModuleInput::const_iterator _formula ) const [inline], [protected]
```

Gets the origins of the passed formula at the given position.

Parameters

_formula	The position of a formula in the passed formulas.

Returns

The origins of the passed formula at the given position.

0.15.308.4.53 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.

0.15.308.4.54 hasLemmas() `bool smtrat::Module::hasLemmas ( ) [inline]`

Checks whether this module or any of its backends provides any lemmas.

0.15.308.4.55 hasValidInfeasibleSubset() `bool smtrat::Module::hasValidInfeasibleSubset ( ) const`

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.308.4.56 id() `std::size_t smtrat::Module::id ( ) const [inline]`

Returns

A unique ID to identify this module instance.

0.15.308.4.57 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets ( ) const [inline]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.308.4.58 inform() `bool smtrat::Module::inform ( const FormulaT & _constraint )`

Informs the module about the given constraint.
It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

Parameters

_constraint	The constraint to inform about.

Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

0.15.308.4.59 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.

0.15.308.4.60 informCore()

`virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.308.4.61 informedConstraints()

`const carl::FastSet<FormulaT>& smtrat::Module::informedConstraints ( ) const [inline]`

Returns

The position of the first constraint of which no backend has been informed about.

0.15.308.4.62 init()

`void smtrat::Module::init ( ) [virtual]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not be called twice and only before the first runBackends call.
Reimplemented in smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntBlastModule< Settings >, smtrat::FPPModule< Settings >, smtrat::CurryModule< Settings >, and smtrat::BVModule< Settings >.

0.15.308.4.63 is_minimizing()

`bool smtrat::Module::is_minimizing ( ) const [inline]`

0.15.308.4.64 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.

0.15.308.4.65 isPreprocessor() `bool smtrat::Module::isPreprocessor ( ) const [inline]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.308.4.66 lemmas() `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.308.4.67 merge()

```
std::vector< FormulaT > smtrat::Module::merge (
    const std::vector< FormulaT > & _vecSetA,
    const std::vector< FormulaT > & _vecSetB ) const [protected]
```

Merges the two vectors of sets into the first one.
({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

Parameters

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns

The vector being the two given vectors merged.

0.15.308.4.68 model() `const Model& smtrat::Module::model ( ) const [inline]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.308.4.69 modelsDisjoint()

```
bool smtrat::Module::modelsDisjoint (
    const Model & _modelA,
    const Model & _modelB ) [static], [protected]
```

Checks whether there is no variable assigned by both models.

Parameters

_modelA	The first model to check for.
_modelB	The second model to check for.

Returns

true, if there is no variable assigned by both models; false, otherwise.

0.15.308.4.70 moduleName() `virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual]`

Returns

The name of the given module type as name.

Reimplemented in smtrat::VSModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::STropModule< Settings >, smtrat::SplitSOSModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntEqModule< Settings >, smtrat::IntBlastModule< Settings >, smtrat::IncWidthModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::GBModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::CurryModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::BVModule< Settings >, and smtrat::BEModule< Settings >.

0.15.308.4.71 objective() `carl::Variable smtrat::Module::objective ( ) const [inline]`

0.15.308.4.72 originInReceivedFormula() `bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected]`

0.15.308.4.73 passedFormulaBegin() `ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.308.4.74 passedFormulaEnd() `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected]`

Returns

An iterator to the end of the passed formula.

0.15.308.4.75 pPassedFormula() `const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.308.4.76 pReceivedFormula() `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline]`

Returns

A pointer to the formula passed to this module.

0.15.308.4.77 print()

```
void smtrat::Module::print (
    const std::string & _initiation = "***" ) const
```

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.308.4.78 printAllModels()

```
void smtrat::Module::printAllModels (
    std::ostream & _out = std::cout )
```

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.308.4.79 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.308.4.80 printModel()

```
void smtrat::Module::printModel (
    std::ostream & _out = std::cout ) const
```

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.308.4.81 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.308.4.82 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.308.4.83 probablyLooping()

```
bool smtrat::Module::probablyLooping (
    const typename Poly::PolyType & _branchingPolynomial,
    const Rational & _branchingValue ) const [protected]
```

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.308.4.84 receivedFormulaChecked()

```
void smtrat::Module::receivedFormulaChecked ( ) [inline]
```

Notifies that the received formulas has been checked.

0.15.308.4.85 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩
InfeasibleSubset (
    ModuleInput::const_iterator _subformula ) [inline], [protected]
```

Stores an infeasible subset consisting only of the given received formula.

0.15.308.4.86 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const [inline]
```

0.15.308.4.87 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.

0.15.308.4.88 removeCore()

```
virtual void smtrat::Module::removeCore (
    [[maybe_unused] ] 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.

0.15.308.4.89 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin (
            ModuleInput::iterator _formula,
            const FormulaT & _origin ) [inline], [protected]
```

0.15.308.4.90 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected]
```

0.15.308.4.91 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.308.4.92 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline]
```

Returns

A reference to the formula passed to this module.

0.15.308.4.93 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual]
```

Reimplemented in smtrat::PModule.

0.15.308.4.94 runBackends() [2/2]

```
Answer smtrat::Module::runBackends (
            bool _final,
            bool _full,
            carl::Variable _objective ) [protected], [virtual]
```

Runs the backend solvers on the passed formula.

Parameters

_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.

0.15.308.4.95 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.

0.15.308.4.96 setThreadPriority() `void smtrat::Module::setThreadPriority ( thread_priority _threadPriority ) [inline]`

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.308.4.97 solverState() `Answer smtrat::Module::solverState ( ) const [inline]`

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.308.4.98 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 !=.

0.15.308.4.99 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.308.4.100 updateAllModels() `void smtrat::Module::updateAllModels ( ) [virtual]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.308.4.101 updateLemmas() `void smtrat::Module::updateLemmas ( )`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.308.4.102 updateModel() `void smtrat::Module::updateModel ( ) const [virtual]`

Updates the model, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::PModule, smtrat::VSModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::STropModule< Settings >, smtrat::SATModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::MCBModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntEqModule< Settings >, smtrat::IntBlastModule< Settings >, smtrat::IncWidthModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::GBPPModule< Settings >, smtrat::FPPModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::ESModule< Settings >, smtrat::CurryModule< Settings >, smtrat::CubeLIAModule< Se
smtrat::CSplitModule< Settings >, smtrat::CoveringNGModule< Settings >, and smtrat::BVModule< Settings >.

0.15.308.4.103 usedBackends() `const std::vector<Module*>& smtrat::Module::usedBackends ( ) const [inline]`

Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.308.5 Field Documentation

0.15.308.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected]`

The backends of this module which have been used.

0.15.308.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected]`

Stores all satisfying assignments.

0.15.308.5.3 mBackendsFoundAnswer `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.308.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected]`

Stores the constraints which the backends must be informed about.

0.15.308.5.5 mFinalCheck `bool smtrat::Module::mFinalCheck [protected]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.308.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static]`

The beginning of the cyclic buffer storing the last branches.

0.15.308.5.7 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.

0.15.308.5.8 mFirstUncheckedReceivedSubformula `ModuleInput::const_iterator smtrat::Module↩ ::mFirstUncheckedReceivedSubformula [protected]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.308.5.9 mFoundAnswer `Conditionals smtrat::Module::mFoundAnswer [protected]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.308.5.10 mFullCheck `bool smtrat::Module::mFullCheck [protected]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.308.5.11 mInfeasibleSubsets `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected]`

Stores the infeasible subsets.

0.15.308.5.12 mInformedConstraints `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.308.5.13 mLastBranches `std::vector< Branching > smtrat::Module::mLastBranches = std↩ ::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static]`

Stores the last branches in a cycle buffer.

0.15.308.5.14 mLemmas `std::vector<Lemma> smtrat::Module::mLemmas [protected]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.308.5.15 mModel `Model smtrat::Module::mModel [mutable], [protected]`

Stores the assignment of the current satisfiable result, if existent.

0.15.308.5.16 mModelComputed `bool smtrat::Module::mModelComputed [mutable], [protected]`

True, if the model has already been computed.

0.15.308.5.17 mNumOfBranchVarsToStore `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.308.5.18 mObjectiveVariable `carl::Variable smtrat::Module::mObjectiveVariable [protected]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.308.5.19 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static]`

Reusable splitting variables.

0.15.308.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected]`

A reference to the manager.

0.15.308.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.308.5.22 mSolverState `std::atomic<Answer> smtrat::Module::mSolverState [protected]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.308.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected]`

0.15.308.5.24 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).

0.15.308.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected]`

Maps variables to the number of their occurrences.

0.15.309 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

0.15.309.1 Constructor & Destructor Documentation

0.15.309.1.1 ModuleFactory() `template<typename Module > smtrat::ModuleFactory< Module >::ModuleFactory ( ) [inline]`

0.15.309.2 Member Function Documentation

0.15.309.2.1 create()

```
template<typename Module >
Module* smtrat::ModuleFactory< Module >::create (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager *const _manager )  [inline], [virtual]
```

Implements smtrat::AbstractModuleFactory.

0.15.309.2.2 moduleName()

```
template<typename Module >
std::string smtrat::ModuleFactory< Module >::moduleName ( ) const  [inline], [virtual]
```

Implements smtrat::AbstractModuleFactory.

0.15.310 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 is_constraint_conjunction () const
- bool isConstraintLiteralConjunction () const
- bool is_real_constraint_conjunction () const
- bool isRealConstraintLiteralConjunction () const
- bool is_integer_constraint_conjunction () const
- bool isIntegerConstraintLiteralConjunction () const
- bool containsBitVectorConstraints () const
- bool containsBooleanVariables () const
- bool containsRealVariables () const
- bool containsIntegerVariables () const
- bool containsUninterpretedEquations () const
- bool is_only_propositional () const
- unsigned satisfiedBy (const Model &_assignment) const
- unsigned satisfiedBy (const RationalAssignment &_assignment) const
- const auto & back () const
- iterator begin ()
- iterator end ()
- const_iterator begin () const
- const_iterator end () const
- auto rbegin () const
- auto rend () const
- bool empty () const
- size_t size () const

- iterator find (const FormulaT &_formula)
- const_iterator find (const FormulaT &_formula) const
- iterator find (const_iterator _hint, const FormulaT &_formula)
- const_iterator find (const_iterator _hint, const FormulaT &_formula) const
- bool contains (const FormulaT &_subformula) const
- void updateProperties ()

  Updates all properties of the formula underlying this module input.
- void gatherVariables (carl::carlVariables &vars) const
- operator FormulaT () const
- void addOrigin (iterator _formula, const FormulaT &_origin)
- iterator erase (iterator _formula)
- void clearOrigins (iterator _formula)
- bool removeOrigin (iterator _formula, const FormulaT &_origin)
- bool removeOrigins (iterator _formula, const std::shared_ptr< FormulasT > &_origins)
- std::pair< iterator, bool > add (const FormulaT &_formula, bool _mightBeConjunction=true)
- std::pair< iterator, bool > add (const FormulaT &_formula, const FormulaT &_origins, bool _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

0.15.310.1 Detailed Description

The input formula a module has to consider for it's satisfiability check.
It is a list of formulas and semantically considered as their conjunction.

0.15.310.2 Member Typedef Documentation

0.15.310.2.1 const_iterator `typedef super::const_iterator smtrat::ModuleInput::const_iterator`

Passing through the list::const_iterator.

0.15.310.2.2 iterator `typedef super::iterator smtrat::ModuleInput::iterator`

Passing through the list::iterator.

0.15.310.3 Constructor & Destructor Documentation

0.15.310.3.1 ModuleInput() `smtrat::ModuleInput::ModuleInput ( ) [inline]`

Constructs a module input.

0.15.310.4 Member Function Documentation

0.15.310.4.1 add() [1/4]

```
std::pair<iterator,bool> smtrat::ModuleInput::add (
    const FormulaT & _formula,
    bool _hasSingleOrigin,
    const FormulaT & _origin,
    const std::shared_ptr< FormulasT > & _origins,
    bool _mightBeConjunction = true )
```

0.15.310.4.2 add() [2/4]

```
std::pair<iterator,bool> smtrat::ModuleInput::add (
    const FormulaT & _formula,
    bool _mightBeConjunction = true ) [inline]
```

0.15.310.4.3 add() [3/4]

```
std::pair<iterator,bool> smtrat::ModuleInput::add (
    const FormulaT & _formula,
    const FormulaT & _origins,
    bool _mightBeConjunction = true ) [inline]
```

0.15.310.4.4 add() [4/4]

```
std::pair<iterator,bool> smtrat::ModuleInput::add (
    const FormulaT & _formula,
    const std::shared_ptr< FormulasT > & _origins,
    bool _mightBeConjunction = true ) [inline]
```

0.15.310.4.5 addOrigin()

```
void smtrat::ModuleInput::addOrigin (
    iterator _formula,
    const FormulaT & _origin ) [inline]
```

0.15.310.4.6 back()

```
const auto& smtrat::ModuleInput::back ( ) const [inline]
```

0.15.310.4.7 begin() [1/2]

```
iterator smtrat::ModuleInput::begin ( ) [inline]
```

0.15.310.4.8 begin() [2/2]

```
const_iterator smtrat::ModuleInput::begin ( ) const [inline]
```

0.15.310.4.9 clearOrigins()

```
void smtrat::ModuleInput::clearOrigins (
    iterator _formula ) [inline]
```

0.15.310.4.10 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.

0.15.310.4.11 containsBitVectorConstraints() `bool smtrat::ModuleInput::containsBitVectorConstraints ( ) const [inline]`

Returns

true, if this formula contains bit vector constraints; false, otherwise.

0.15.310.4.12 containsBooleanVariables() `bool smtrat::ModuleInput::containsBooleanVariables ( ) const [inline]`

Returns

true, if this formula contains Boolean variables; false, otherwise.

0.15.310.4.13 containsIntegerVariables() `bool smtrat::ModuleInput::containsIntegerVariables ( ) const [inline]`

Returns

true, if this formula contains Boolean variables; false, otherwise.

0.15.310.4.14 containsRealVariables() `bool smtrat::ModuleInput::containsRealVariables ( ) const [inline]`

Returns

true, if this formula contains Boolean variables; false, otherwise.

0.15.310.4.15 containsUninterpretedEquations() `bool smtrat::ModuleInput::containsUninterpreted↩ Equations ( ) const [inline]`

Returns

true, if this formula contains uninterpreted equations; false, otherwise.

0.15.310.4.16 empty() `bool smtrat::ModuleInput::empty ( ) const [inline]`

0.15.310.4.17 end() [1/2] `iterator smtrat::ModuleInput::end ( ) [inline]`

0.15.310.4.18 end() [2/2] `const_iterator smtrat::ModuleInput::end ( ) const [inline]`

0.15.310.4.19 erase() `ModuleInput::iterator smtrat::ModuleInput::erase ( iterator _formula )`

0.15.310.4.20 find() [1/4] `ModuleInput::iterator smtrat::ModuleInput::find ( const FormulaT & _formula )`

0.15.310.4.21 find() [2/4]

```
ModuleInput::const_iterator smtrat::ModuleInput::find (
const FormulaT & _formula ) const
```

0.15.310.4.22 find() [3/4]

```
ModuleInput::iterator smtrat::ModuleInput::find (
const_iterator _hint,
const FormulaT & _formula )
```

0.15.310.4.23 find() [4/4]

```
ModuleInput::const_iterator smtrat::ModuleInput::find (
const_iterator _hint,
const FormulaT & _formula ) const
```

0.15.310.4.24 gatherVariables()

```
void smtrat::ModuleInput::gatherVariables (
carl::carlVariables & vars ) const [inline]
```

0.15.310.4.25 is_constraint_conjunction()

```
bool smtrat::ModuleInput::is_constraint_conjunction ( )
const [inline]
```

Returns

true, if this formula is a conjunction of constraints; false, otherwise.

0.15.310.4.26 is_integer_constraint_conjunction()

```
bool smtrat::ModuleInput::is_integer_constraint↩
_conjunction ( ) const [inline]
```

Returns

true, if this formula is a conjunction of integer constraints; false, otherwise.

0.15.310.4.27 is_only_propositional()

```
bool smtrat::ModuleInput::is_only_propositional ( ) const
[inline]
```

Returns

true, if this formula is propositional; false, otherwise.

0.15.310.4.28 is_real_constraint_conjunction()

```
bool smtrat::ModuleInput::is_real_constraint_↩
conjunction ( ) const [inline]
```

Returns

true, if this formula is a conjunction of real constraints; false, otherwise.

0.15.310.4.29 isConstraintLiteralConjunction()

```
bool smtrat::ModuleInput::isConstraintLiteral↩
Conjunction ( ) const [inline]
```

Returns

true, if this formula is a conjunction of literals of constraints; false, otherwise.

0.15.310.4.30 isIntegerConstraintLiteralConjunction()

`bool smtrat::ModuleInput::isIntegerConstraint↩ LiteralConjunction ( ) const [inline]`

Returns

true, if this formula is a conjunction of literals of integer constraints; false, otherwise.

0.15.310.4.31 isRealConstraintLiteralConjunction()

`bool smtrat::ModuleInput::isRealConstraint↩ LiteralConjunction ( ) const [inline]`

Returns

true, if this formula is a conjunction of literals of real constraints; false, otherwise.

0.15.310.4.32 operator FormulaT()

`smtrat::ModuleInput::operator FormulaT ( ) const [inline], [explicit]`

0.15.310.4.33 properties()

`carl::Condition smtrat::ModuleInput::properties ( ) const [inline]`

Returns

All known properties of the underlying formula of this module input.

0.15.310.4.34 rbegin()

`auto smtrat::ModuleInput::rbegin ( ) const [inline]`

0.15.310.4.35 removeOrigin()

```
bool smtrat::ModuleInput::removeOrigin (
    iterator _formula,
    const FormulaT & _origin )
```

0.15.310.4.36 removeOrigins()

```
bool smtrat::ModuleInput::removeOrigins (
    iterator _formula,
    const std::shared_ptr< FormulasT > & _origins )
```

0.15.310.4.37 rend()

`auto smtrat::ModuleInput::rend ( ) const [inline]`

0.15.310.4.38 satisfiedBy() [1/2]

```
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.

0.15.310.4.39 satisfiedBy() [2/2] `unsigned smtrat::ModuleInput::satisfiedBy ( const RationalAssignment & _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.

0.15.310.4.40 size() `size_t smtrat::ModuleInput::size ( void ) const [inline]`

0.15.310.4.41 updateProperties() `void smtrat::ModuleInput::updateProperties ( )`

Updates all properties of the formula underlying this module input.

0.15.310.5 Friends And Related Function Documentation

0.15.310.5.1 Manager `friend class Manager [friend]`

0.15.310.5.2 Module `friend class Module [friend]`

0.15.311 smtrat::settings::ModuleSettings Struct Reference

`#include <Settings.h>`

Public Member Functions

- ModuleSettings ()
- void set_callbacks (std::function< bool(const std::string &)> callback_has, std::function< const std::string &(const std::string &)> callback_get)
- template<typename T >
  T get (const std::string &key, T default_value) const

Data Fields

- std::vector< std::string > parameters

0.15.311.1 Constructor & Destructor Documentation

0.15.311.1.1 ModuleSettings() `smtrat::settings::ModuleSettings::ModuleSettings ( ) [inline]`

0.15.311.2 Member Function Documentation

0.15.311.2.1 get()

```
template<typename T >
T smtrat::settings::ModuleSettings::get (
            const std::string & key,
            T default_value ) const [inline]
```

0.15.311.2.2 set_callbacks()

```
void smtrat::settings::ModuleSettings::set_callbacks (
            std::function< bool(const std::string &)> callback_has,
            std::function< const std::string &(const std::string &)> callback_get ) [inline]
```

0.15.311.3 Field Documentation

0.15.311.3.1 parameters

```
std::vector<std::string> smtrat::settings::ModuleSettings::parameters
```

0.15.312 smtrat::Module::ModuleStatistics Struct Reference

```
#include <Module.h>
```

Public Member Functions

- void start_add ()
- void start_check ()
- void start_remove ()
- void pause_all ()
- void continue_all ()
- void stop_add ()
- void stop_check ()
- void stop_remove ()

0.15.312.1 Member Function Documentation

0.15.312.1.1 continue_all()

```
void smtrat::Module::ModuleStatistics::continue_all ( ) [inline]
```

0.15.312.1.2 pause_all()

```
void smtrat::Module::ModuleStatistics::pause_all ( ) [inline]
```

0.15.312.1.3 start_add()

```
void smtrat::Module::ModuleStatistics::start_add ( ) [inline]
```

0.15.312.1.4 start_check()

```
void smtrat::Module::ModuleStatistics::start_check ( ) [inline]
```

0.15.312.1.5 start_remove()

```
void smtrat::Module::ModuleStatistics::start_remove ( ) [inline]
```

0.15.312.1.6 stop_add()

```
void smtrat::Module::ModuleStatistics::stop_add ( ) [inline]
```

0.15.312.1.7 stop_check()

```
void smtrat::Module::ModuleStatistics::stop_check ( ) [inline]
```

0.15.312.1.8 stop_remove() `void smtrat::Module::ModuleStatistics::stop_remove ( ) [inline]`

0.15.313 smtrat::subtropical::Moment Struct Reference

Represents the normal vector component and the sign variable assigned to a variable in an original constraint.
`#include <Subtropical.h>`

Public Member Functions

- Moment ()

Data Fields

- const carl::Variable normal_vector

  Normal vector component of the separating hyperplane.
- const carl::Variable sign_variant

  Boolean variable representing the sign variant.

0.15.313.1 Detailed Description

Represents the normal vector component and the sign variable assigned to a variable in an original constraint.

0.15.313.2 Constructor & Destructor Documentation

0.15.313.2.1 Moment() `smtrat::subtropical::Moment::Moment ( ) [inline]`

0.15.313.3 Field Documentation

0.15.313.3.1 normal_vector `const carl::Variable smtrat::subtropical::Moment::normal_vector`

Normal vector component of the separating hyperplane.

0.15.313.3.2 sign_variant `const carl::Variable smtrat::subtropical::Moment::sign_variant`

Boolean variable representing the sign variant.

0.15.314 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 >

0.15.314.1 Member Function Documentation

0.15.314.1.1 getMMonomialMapping()

`const MonomialMap& smtrat::MonomialMappingByVariablePool::getMMonomialMapping ( ) const [inline]`

0.15.314.1.2 insertMonomialMapping()

```
void smtrat::MonomialMappingByVariablePool::insertMonomialMapping (
    carl::Variable variable,
    carl::Monomial::Arg monomial ) [inline]
```

0.15.314.1.3 isNull()

```
bool smtrat::MonomialMappingByVariablePool::isNull (
    carl::Variable variable ) [inline]
```

0.15.314.1.4 monomial()

```
carl::Monomial::Arg smtrat::MonomialMappingByVariablePool::monomial (
    carl::Variable variable ) [inline]
```

0.15.314.1.5 variable()

```
carl::Variable smtrat::MonomialMappingByVariablePool::variable (
    carl::Monomial::Arg monomial ) [inline]
```

0.15.314.2 Friends And Related Function Documentation

0.15.314.2.1 carl::Singleton< MonomialMappingByVariablePool >

`friend class carl::Singleton< MonomialMappingByVariablePool > [friend]`

0.15.315 smtrat::expression::NaryExpression Struct Reference

`#include <ExpressionContent.h>`

Public Member Functions

- NaryExpression (NaryType _type, Expressions &&_expressions)
- NaryExpression (NaryType _type, const std::initializer_list< Expression > &_expressions)
- void normalize ()

Data Fields

- NaryType type
- Expressions expressions

0.15.315.1 Constructor & Destructor Documentation

0.15.315.1.1 NaryExpression() [1/2]

```
smtrat::expression::NaryExpression::NaryExpression (
    NaryType _type,
    Expressions && _expressions ) [inline]
```

0.15.315.1.2 NaryExpression() [2/2]

```
smtrat::expression::NaryExpression::NaryExpression (
    NaryType _type,
    const std::initializer_list< Expression > & _expressions ) [inline]
```

0.15.315.2 Member Function Documentation

0.15.315.2.1 normalize() `void smtrat::expression::NaryExpression::normalize ( ) [inline]`

0.15.315.3 Field Documentation

0.15.315.3.1 expressions `Expressions smtrat::expression::NaryExpression::expressions`

0.15.315.3.2 type `NaryType smtrat::expression::NaryExpression::type`

0.15.316 smtrat::expression::simplifier::NegationSimplifier Struct Reference

`#include <NegationSimplifier.h>`

Public Member Functions

- const ExpressionContent ∗ simplify (const UnaryExpression &expr) const
- const ExpressionContent ∗ operator() (const carl::Variable &expr) const
- const ExpressionContent ∗ operator() (const ITEExpression &expr) const
- const ExpressionContent ∗ operator() (const QuantifierExpression &expr) const
- const ExpressionContent ∗ operator() (const UnaryExpression &expr) const
- const ExpressionContent ∗ operator() (const BinaryExpression &expr) const
- const ExpressionContent ∗ operator() (const NaryExpression &expr) const
- const ExpressionContent ∗ operator() (const ExpressionContent ∗_ec) const

Protected Member Functions

- virtual const ExpressionContent ∗ simplify (const carl::Variable &) const
- virtual const ExpressionContent ∗ simplify (const ITEExpression &) const
- virtual const ExpressionContent ∗ simplify (const QuantifierExpression &) const
- virtual const ExpressionContent ∗ simplify (const BinaryExpression &) const
- virtual const ExpressionContent ∗ simplify (const NaryExpression &) const

0.15.316.1 Member Function Documentation

0.15.316.1.1 operator()() [1/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const BinaryExpression & expr ) const [inline], [inherited]
```

0.15.316.1.2 operator()() [2/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const carl::Variable & expr ) const [inline], [inherited]
```

0.15.316.1.3 operator()() [3/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const ExpressionContent * _ec ) const [inline], [inherited]
```

0.15.316.1.4 operator()() [4/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
    const ITEExpression & expr ) const [inline], [inherited]
```

0.15.316.1.5 operator()() [5/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
    const NaryExpression & expr ) const [inline], [inherited]
```

0.15.316.1.6 operator()() [6/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
    const QuantifierExpression & expr ) const [inline], [inherited]
```

0.15.316.1.7 operator()() [7/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
    const UnaryExpression & expr ) const [inline], [inherited]
```

0.15.316.1.8 simplify() [1/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
    const BinaryExpression & ) const [inline], [protected], [virtual], [inherited]
```

0.15.316.1.9 simplify() [2/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
    const carl::Variable & ) const [inline], [protected], [virtual], [inherited]
```

0.15.316.1.10 simplify() [3/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
    const ITEExpression & ) const [inline], [protected], [virtual], [inherited]
```

0.15.316.1.11 simplify() [4/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
    const NaryExpression & ) const [inline], [protected], [virtual], [inherited]
```

Reimplemented in smtrat::expression::simplifier::SingletonSimplifier, smtrat::expression::simplifier::MergeSimplifier, and smtrat::expression::simplifier::DuplicateSimplifier.

0.15.316.1.12 simplify() [5/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
    const QuantifierExpression & ) const [inline], [protected], [virtual], [inherited]
```

0.15.316.1.13 simplify() [6/6]

```
const ExpressionContent* smtrat::expression::simplifier::NegationSimplifier::simplify (
    const UnaryExpression & expr ) const [inline], [virtual]
```

Reimplemented from smtrat::expression::simplifier::BaseSimplifier.

0.15.317 smtrat::NewCADModule< Settings > Class Template Reference

```
#include <NewCADModule.h>
```

Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- NewCADModule (const ModuleInput ∗_formula, Conditionals &_conditionals, Manager ∗_manager=nullptr)
- ∼NewCADModule ()
- bool informCore (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void init ()

  Informs all backends about the so far encountered constraints, which have not yet been communicated.
- bool addCore (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- void removeCore (ModuleInput::const_iterator _subformula)

  Removes the subformula of the received formula at the given position to the considered ones of this module.
- void updateModel () const

  Updates the current assignment into the model.
- Answer checkCore ()

  Checks the received formula for consistency.
- bool inform (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void 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 bool informCore ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.
- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)
- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.

- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

  Copies the infeasible subsets of the passed formula.
- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

Gets all InformationRelevantFormulas.

- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.317.1 Member Typedef Documentation

0.15.317.1.1 SettingsType `template<typename Settings >`

`typedef Settings smtrat::NewCADModule< Settings >::SettingsType`

0.15.317.2 Member Enumeration Documentation

0.15.317.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

0.15.317.3 Constructor & Destructor Documentation

0.15.317.3.1 NewCADModule() `template<typename Settings >`

```
smtrat::NewCADModule< Settings >::NewCADModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

0.15.317.3.2 ~NewCADModule() `template<typename Settings >`

`smtrat::NewCADModule< Settings >::~NewCADModule ( )`

0.15.317.4 Member Function Documentation

0.15.317.4.1 add()

```
bool smtrat::Module::add (
            ModuleInput::const_iterator _subformula ) [inherited]
```

The module has to take the given sub-formula of the received formula into account.

Parameters

_subformula	The sub-formula to take additionally into account.

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

0.15.317.4.2 addConstraintToInform()

```
void smtrat::Module::addConstraintToInform (
            const FormulaT & _constraint ) [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

0.15.317.4.3 addCore() [1/2]

```
virtual bool smtrat::Module::addCore (
    [[maybe_unused] ] 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.

0.15.317.4.4 addCore() [2/2]

```
template<typename Settings >
bool smtrat::NewCADModule< Settings >::addCore (
    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.

0.15.317.4.5 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula
(
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.317.4.6 addLemma()

```
void smtrat::Module::addLemma (
    const FormulaT & _lemma,
    const LemmaType & _lt = LemmaType::NORMAL,
    const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline],
[inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.317.4.7 addOrigin()

```
void smtrat::Module::addOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.317.4.8 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.317.4.9 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.317.4.10 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.317.4.11 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.

0.15.317.4.12 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns

All satisfying assignments, if existent.

0.15.317.4.13 anAnswerFound()

```
bool smtrat::Module::anAnswerFound ( ) const [inline], [protected],
[inherited]
```

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.317.4.14 answerFound()

```
const smtrat::Conditionals& smtrat::Module::answerFound ( ) const
[inline], [inherited]
```

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.317.4.15 backendsModel() `const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.317.4.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.317.4.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.317.4.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.317.4.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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.

Parameters

_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

0.15.317.4.20 check()

```
Answer smtrat::Module::check (
    bool _final = false,
    bool _full = true,
    carl::Variable _objective = carl::Variable::NO_VARIABLE ) [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.317.4.21 checkCore()

```
template<typename 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.

0.15.317.4.22 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.

0.15.317.4.23 checkModel() `unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]`

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.317.4.24 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.317.4.25 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

0.15.317.4.26 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

0.15.317.4.27 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

0.15.317.4.28 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.317.4.29 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

0.15.317.4.30 collectOrigins() [2/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.317.4.31 collectTheoryPropagations()

`void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

0.15.317.4.32 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ToInform ( ) const [inline], [inherited]`

Returns

The constraints which the backends must be informed about.

0.15.317.4.33 currentlySatisfied()

`virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.317.4.34 currentlySatisfiedByBackend()

`unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.317.4.35 deinform()

`void smtrat::Module::deinform ( const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.317.4.36 deinformCore()

`virtual void smtrat::Module::deinformCore ( [[maybe_unused] ] 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.

0.15.317.4.37 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

0.15.317.4.38 eraseSubformulaFromPassedFormula()

`ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula ( ModuleInput::iterator _subformula, bool _ignoreOrigins = false ) [protected], [virtual], [inherited]`

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_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 >.

0.15.317.4.39 excludeNotReceivedVariablesFromModel()

`void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

0.15.317.4.40 findBestOrigin()

`std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin ( const std::vector< FormulaT > & _origins ) const [protected], [inherited]`

Parameters

_origins	

Returns

0.15.317.4.41 firstSubformulaToPass() `ModuleInput::const_iterator smtrat::Module::firstSubformula↩ ToPass ( ) const [inline], [inherited]`

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.317.4.42 firstUncheckedReceivedSubformula() `ModuleInput::const_iterator smtrat::Module↩ ::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.317.4.43 freeSplittingVariable() `static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.317.4.44 generateTrivialInfeasibleSubset() `void smtrat::Module::generateTrivialInfeasible↩ Subset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.317.4.45 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.317.4.46 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.317.4.47 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.317.4.48 getInfeasibleSubsets() [2/2] `std::vector< FormulaSetT > smtrat::Module::get↩ InfeasibleSubsets ( 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.

0.15.317.4.49 getInformationRelevantFormulas() `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.317.4.50 getModelEqualities() `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.317.4.51 getOrigins() [1/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.317.4.52 getOrigins() [2/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.317.4.53 getOrigins() [3/3] `const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]`

Gets the origins of the passed formula at the given position.

Parameters

_formula	The position of a formula in the passed formulas.

Returns

The origins of the passed formula at the given position.

0.15.317.4.54 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.

0.15.317.4.55 hasLemmas() `bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

0.15.317.4.56 hasValidInfeasibleSubset() `bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]`

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.317.4.57 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.317.4.58 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.317.4.59 inform() `bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.
It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

Parameters

_constraint	The constraint to inform about.

Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

0.15.317.4.60 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.

0.15.317.4.61 informCore() [1/2]

`virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.317.4.62 informCore() [2/2]

`template<typename Settings > bool smtrat::NewCADModule< Settings >::informCore ( 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.

0.15.317.4.63 informedConstraints()

`const carl::FastSet<FormulaT>& smtrat::Module::informedConstraints ( ) const [inline], [inherited]`

Returns

The position of the first constraint of which no backend has been informed about.

0.15.317.4.64 init() `template<typename Settings >`
`void smtrat::NewCADModule< Settings >::init ( ) [virtual]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not and will not be called more than once and only before the first runBackends call.
Reimplemented from smtrat::Module.

0.15.317.4.65 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.317.4.66 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.

0.15.317.4.67 isPreprocessor() `bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.317.4.68 lemmas() `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.317.4.69 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.

0.15.317.4.70 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.317.4.71 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.

0.15.317.4.72 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.

0.15.317.4.73 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

0.15.317.4.74 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.317.4.75 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.317.4.76 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.317.4.77 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.317.4.78 pReceivedFormula() `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.317.4.79 print() `void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.317.4.80 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.

0.15.317.4.81 printInfeasibleSubsets() `void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.317.4.82 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.

0.15.317.4.83 printPassedFormula() `void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.317.4.84 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.317.4.85 probablyLooping()

`bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.317.4.86 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.317.4.87 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

0.15.317.4.88 receivedVariable()

`bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

0.15.317.4.89 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.

0.15.317.4.90 removeCore() [1/2]

```
virtual void smtrat::Module::removeCore (
    [[maybe_unused] ] 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.

0.15.317.4.91 removeCore() [2/2]

```
template<typename Settings >
void smtrat::NewCADModule< Settings >::removeCore (
    ModuleInput::const_iterator _subformula )
```

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.

0.15.317.4.92 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

0.15.317.4.93 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

0.15.317.4.94 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.317.4.95 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to this module.

0.15.317.4.96 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( ) [inline],
[protected], [virtual], [inherited]
```

Reimplemented in smtrat::PModule.

0.15.317.4.97 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.

0.15.317.4.98 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.

0.15.317.4.99 setThreadPriority()

```
void smtrat::Module::setThreadPriority (
    thread_priority _threadPriority ) [inline], [inherited]
```

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.317.4.100 solverState()

```
Answer smtrat::Module::solverState ( ) const [inline], [inherited]
```

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.317.4.101 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 !=.

0.15.317.4.102 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.317.4.103 updateAllModels() `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.317.4.104 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.317.4.105 updateModel() `template<typename Settings > void smtrat::NewCADModule< Settings >::updateModel ( ) const [virtual]`

Updates the current assignment into the model.
Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.
Reimplemented from smtrat::Module.

0.15.317.4.106 usedBackends() `const std::vector<Module*>& smtrat::Module::usedBackends ( ) const [inline], [inherited]`

Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.317.5 Field Documentation

0.15.317.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.317.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.317.5.3 mBackendsFoundAnswer `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.317.5.4 mConstraintsToInform

`carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.317.5.5 mFinalCheck

`bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.317.5.6 mFirstPosInLastBranches

`std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.317.5.7 mFirstSubformulaToPass

`ModuleInput::iterator smtrat::Module::mFirstSubformulaTo↩ Pass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.317.5.8 mFirstUncheckedReceivedSubformula

`ModuleInput::const_iterator smtrat::Module↩ ::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.317.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.317.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.317.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.317.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.317.5.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std↩ ::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.317.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.317.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.317.5.16 mModelComputed `bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.317.5.17 mNumOfBranchVarsToStore `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.317.5.18 mObjectiveVariable `carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.317.5.19 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

0.15.317.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.317.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.317.5.22 mSolverState `std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.317.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

0.15.317.5.24 mUsedBackends `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.317.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.318 smtrat::NewCoveringModule< Settings > Class Template Reference

`#include <NewCoveringModule.h>`

Public Types

- using SettingsType = Settings
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- NewCoveringModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=nullptr)
- ~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.
- void processAnswer ()

  Processes the current answer, i.e.
- 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.
- std::optional< Answer > doBacktracking ()

  The actual backtracking algorithm, which is called when we have constraints to add and no constraints to remove.
- std::optional< Answer > doIncremental ()

  The actual incremental algorithm, which is called when we have constraints to add and no constraints to remove.
- std::optional< Answer > doIncrementalAndBacktracking ()

  Algorithms which contains backtracking and incremental checking, its called when we have constraints to add and constraints to remove.
- bool inform (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void deinform (const FormulaT &_constraint)

  The inverse of informing about a constraint.
- bool add (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- virtual Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)

  Checks the received formula for consistency.
- virtual void remove (ModuleInput::const_iterator _subformula)

  Removes everything related to the given sub-formula of the received formula.
- virtual void updateAllModels ()

  Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
- virtual std::list< std::vector< carl::Variable > > getModelEqualities () const

  Partition the variables from the current model into equivalence classes according to their assigned value.
- unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const
- virtual unsigned currentlySatisfied (const FormulaT &) const
- bool receivedVariable (carl::Variable::Arg _var) const

- Answer solverState () const
- std::size_t 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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.
- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const

- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)
- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

  Copies the infeasible subsets of the passed formula.
- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.

- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &↩ _premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeft↩ Case=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std↩ ::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.

- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.318.1 Member Typedef Documentation

0.15.318.1.1 SettingsType `template<typename Settings >`

```
using smtrat::NewCoveringModule< Settings >::SettingsType = Settings
```

0.15.318.2 Member Enumeration Documentation

0.15.318.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

0.15.318.3 Constructor & Destructor Documentation

0.15.318.3.1 NewCoveringModule() `template<typename Settings >`

```
smtrat::NewCoveringModule< Settings >::NewCoveringModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

0.15.318.3.2 ~NewCoveringModule() `template<typename Settings >`

```
smtrat::NewCoveringModule< Settings >::~NewCoveringModule ( )
```

0.15.318.4 Member Function Documentation

0.15.318.4.1 add() `bool smtrat::Module::add (`

```
            ModuleInput::const_iterator _subformula ) [inherited]
```

The module has to take the given sub-formula of the received formula into account.

Parameters

_subformula	The sub-formula to take additionally into account.

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

0.15.318.4.2 addConstraintsSAT() `template<typename Settings >`

`size_t smtrat::NewCoveringModule< Settings >::addConstraintsSAT ( )`

Adds the given constraint to the backend.

Returns

the lowest level with an unsatisfied new constraint

Note

returns mVariableOrdering.size()+1 if all new constraints are satisfied

If not all new constraints are satisfied the

0.15.318.4.3 addConstraintsUNSAT() `template<typename Settings >`

`void smtrat::NewCoveringModule< Settings >::addConstraintsUNSAT ( )`

Adds the given constraint to the backend.

0.15.318.4.4 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 >.

0.15.318.4.5 addCore() [1/2] `virtual bool smtrat::Module::addCore ( [[maybe_unused] ] 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.

0.15.318.4.6 addCore() [2/2]

```
template<typename Settings >
bool smtrat::NewCoveringModule< Settings >::addCore (
            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.

0.15.318.4.7 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula
(
            const FormulaT & formula )  [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.318.4.8 addLemma()

```
void smtrat::Module::addLemma (
            const FormulaT & _lemma,
            const LemmaType & _lt = LemmaType::NORMAL,
            const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) )  [inline],
[inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.318.4.9 addOrigin()

```
void smtrat::Module::addOrigin (
            ModuleInput::iterator _formula,
            const FormulaT & _origin )  [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.318.4.10 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
```

```
ModuleInput::const_iterator _subformula )  [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.318.4.11 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula )  [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.318.4.12 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula,
            const FormulaT & _origin )  [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.318.4.13 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.

0.15.318.4.14 allModels() `const std::vector<Model>& smtrat::Module::allModels ( ) const [inline], [inherited]`

Returns

All satisfying assignments, if existent.

0.15.318.4.15 anAnswerFound() `bool smtrat::Module::anAnswerFound ( ) const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.318.4.16 answerFound() `const smtrat::Conditionals& smtrat::Module::answerFound ( ) const [inline], [inherited]`

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.318.4.17 backendsModel() `const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.318.4.18 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.318.4.19 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.318.4.20 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline],
[protected], [inherited]
```

0.15.318.4.21 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.318.4.22 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.

0.15.318.4.23 checkCore()

`template<typename Settings >`
`Answer smtrat::NewCoveringModule< Settings >::checkCore ( ) [virtual]`

Checks the received formula for consistency.

Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.318.4.24 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.

0.15.318.4.25 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.

0.15.318.4.26 cleanModel()

`void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.318.4.27 clearLemmas()

`void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

0.15.318.4.28 clearModel()

`void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

0.15.318.4.29 clearModels()

`void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

0.15.318.4.30 clearPassedFormula()

`void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.318.4.31 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

0.15.318.4.32 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.

0.15.318.4.33 collectTheoryPropagations()

`void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

0.15.318.4.34 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns

The constraints which the backends must be informed about.

0.15.318.4.35 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

0.15.318.4.36 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)

0.15.318.4.37 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.

0.15.318.4.38 deinformCore()

```
virtual void smtrat::Module::deinformCore (
    [[maybe_unused] ] 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.

0.15.318.4.39 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

0.15.318.4.40 doBacktracking()

```
template<typename Settings >
std::optional<Answer> smtrat::NewCoveringModule< Settings >::doBacktracking ( )
```

The actual backtracking algorithm, which is called when we have constraints to add and no constraints to remove. Removes all cells which were created as a rest of the constraints which are to be removed if level 0 is still fully covered after removing the constraints UNSAT is returned Otherwise std::nullopt is returned and level 1 is newly sampled and the algorithm starts again from level 1

Returns

std::optional<Answer>, which contains the answer if it can be deduced directly

0.15.318.4.41 doIncremental()

`template<typename Settings >`
`std::optional<Answer> smtrat::NewCoveringModule< Settings >::doIncremental ( )`

The actual incremental algorithm, which is called when we have constraints to add and no constraints to remove. Tests the new constraints with the last satisfying assignment, if the new constraints are also satisfied. If the new constraints are all satisfied SAT is returned accordingly Otherwise std::nullopt is returned and the stored information from the lowest level with an unsatisfied new constraint is removed

Returns

std::optional<Answer>, which contains the answer if it can be deduced directly

0.15.318.4.42 doIncrementalAndBacktracking()

`template<typename Settings >`
`std::optional<Answer> smtrat::NewCoveringModule< Settings >::doIncrementalAndBacktracking ( )`

Algorithms which contains backtracking and incremental checking, its called when we have constraints to add and constraints to remove.

Returns

std::optional<Answer>, which contains the answer if it can be deduced directly

0.15.318.4.43 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 >.

0.15.318.4.44 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceived↩
VariablesFromModel ( ) const  [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

0.15.318.4.45 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::find↩
BestOrigin (
            const std::vector< FormulaT > & _origins ) const  [protected], [inherited]
```

Parameters

_origins	

Returns

0.15.318.4.46 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.

0.15.318.4.47 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.

0.15.318.4.48 freeSplittingVariable() `static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.318.4.49 generateTrivialInfeasibleSubset() `void smtrat::Module::generateTrivialInfeasible↩ Subset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.318.4.50 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.318.4.51 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.318.4.52 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.318.4.53 getInfeasibleSubsets() [2/2] `std::vector< FormulaSetT > smtrat::Module::get↩ InfeasibleSubsets ( 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.

0.15.318.4.54 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.318.4.55 getModelEqualities()

`std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.318.4.56 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.318.4.57 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.318.4.58 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.

0.15.318.4.59 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.

0.15.318.4.60 hasLemmas() `bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

0.15.318.4.61 hasValidInfeasibleSubset() `bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]`

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.318.4.62 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.318.4.63 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.318.4.64 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.

0.15.318.4.65 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.

0.15.318.4.66 informCore() [1/2]

`virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.318.4.67 informCore() [2/2]

`template<typename Settings > bool smtrat::NewCoveringModule< Settings >::informCore ( 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.

0.15.318.4.68 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.

0.15.318.4.69 init() `template<typename 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.

0.15.318.4.70 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.318.4.71 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.

0.15.318.4.72 isPreprocessor() `bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.318.4.73 lemmas() `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.318.4.74 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.

0.15.318.4.75 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.318.4.76 modelsDisjoint()

```
bool smtrat::Module::modelsDisjoint (
    const Model & _modelA,
    const Model & _modelB ) [static], [protected], [inherited]
```

Checks whether there is no variable assigned by both models.

Parameters

_modelA	The first model to check for.
_modelB	The second model to check for.

Returns

true, if there is no variable assigned by both models; false, otherwise.

0.15.318.4.77 moduleName()

```
virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]
```

Returns

The name of the given module type as name.

Reimplemented in smtrat::VSModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::STropModule< Settings >, smtrat::SplitSOSModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntEqModule< Settings >, smtrat::IntBlastModule< Settings >, smtrat::IncWidthModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::GBModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::CurryModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::BVModule< Settings >, and smtrat::BEModule< Settings >.

0.15.318.4.78 objective()

```
carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]
```

0.15.318.4.79 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
    const FormulaT & _origin ) const [protected], [inherited]
```

0.15.318.4.80 passedFormulaBegin()

```
ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]
```

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.318.4.81 passedFormulaEnd()

```
ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]
```

Returns

An iterator to the end of the passed formula.

0.15.318.4.82 pPassedFormula() `const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.318.4.83 pReceivedFormula() `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.318.4.84 print() `void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.318.4.85 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.

0.15.318.4.86 printInfeasibleSubsets() `void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.318.4.87 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.

0.15.318.4.88 printPassedFormula()

`void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.318.4.89 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.318.4.90 probablyLooping()

`bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.318.4.91 processAnswer()

`template<typename Settings > void smtrat::NewCoveringModule< Settings >::processAnswer ( )`

Processes the current answer, i.e.
computes a model or adds the infeasible subset.

0.15.318.4.92 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.318.4.93 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

0.15.318.4.94 receivedVariable() `bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

0.15.318.4.95 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.

0.15.318.4.96 removeConstraintsSAT() `template<typename Settings > void smtrat::NewCoveringModule< Settings >::removeConstraintsSAT ( )`

Removes the given constraints from the backend.

Note

This also removes all stored information about the removed constraints

0.15.318.4.97 removeConstraintsUNSAT() `template<typename Settings > bool smtrat::NewCoveringModule< Settings >::removeConstraintsUNSAT ( )`

Removes the given constraints from the backend Also removes all stored information about the removed constraints.

Returns

True: If the model is still unsat after removing the constraints False: If the model might needs re-computation for the higher levels

0.15.318.4.98 removeCore() [1/2] `virtual void smtrat::Module::removeCore ( [[maybe_unused] ] 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.

0.15.318.4.99 removeCore() [2/2] `template<typename Settings > void smtrat::NewCoveringModule< Settings >::removeCore ( ModuleInput::const_iterator _subformula )`

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.

0.15.318.4.100 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin (
            ModuleInput::iterator _formula,
            const FormulaT & _origin ) [inline], [protected], [inherited]
```

0.15.318.4.101 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]
```

0.15.318.4.102 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.318.4.103 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]
```

Returns

A reference to the formula passed to this module.

0.15.318.4.104 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]
```

Reimplemented in smtrat::PModule.

0.15.318.4.105 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.

0.15.318.4.106 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.

0.15.318.4.107 setThreadPriority()

`void smtrat::Module::setThreadPriority ( thread_priority _threadPriority ) [inline], [inherited]`

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.318.4.108 solverState()

`Answer smtrat::Module::solverState ( ) const [inline], [inherited]`

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.318.4.109 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 !=.

0.15.318.4.110 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.318.4.111 updateAllModels() `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.318.4.112 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.318.4.113 updateModel() `template<typename Settings > void smtrat::NewCoveringModule< Settings >::updateModel ( ) const [virtual]`

Updates the current assignment into the model.
Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.
Reimplemented from smtrat::Module.

0.15.318.4.114 usedBackends() `const std::vector<Module*>& smtrat::Module::usedBackends ( ) const [inline], [inherited]`

Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.318.5 Field Documentation

0.15.318.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.318.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.318.5.3 mBackendsFoundAnswer `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.318.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.318.5.5 mFinalCheck `bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.318.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.318.5.7 mFirstSubformulaToPass

`ModuleInput::iterator smtrat::Module::mFirstSubformulaTo↩Pass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.318.5.8 mFirstUncheckedReceivedSubformula

`ModuleInput::const_iterator smtrat::Module↩::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.318.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.318.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.318.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.318.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.318.5.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std↩::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.318.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.318.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.318.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.318.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.318.5.18 mObjectiveVariable `carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.318.5.19 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]`

Reusable splitting variables.

0.15.318.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.318.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.318.5.22 mSolverState `std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.318.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]`

0.15.318.5.24 mUsedBackends `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.318.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.319 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)

0.15.319.1 Detailed Description

Specification of a compuation node for the SSH backend.

0.15.319.2 Field Documentation

0.15.319.2.1 connections `std::size_t benchmax::ssh::Node::connections`

Number of concurrent connections (default is 1)

0.15.319.2.2 cores `unsigned long benchmax::ssh::Node::cores`

Number of cores we use per connection (default is 1)

0.15.319.2.3 hostname `std::string benchmax::ssh::Node::hostname`

Hostname to connect to.

0.15.319.2.4 password `std::string benchmax::ssh::Node::password`

Password (only used if public key authentication fails).

0.15.319.2.5 port `int benchmax::ssh::Node::port`

Port (default is 22)

0.15.319.2.6 username `std::string benchmax::ssh::Node::username`

Username.

0.15.320 smtrat::covering_ng::formula::node_ds::Node Class Reference

`#include <FormulaEvaluation.h>`

Public Member Functions

- Node (const Node &c)
- template<typename C >
  requires (!std::same_as< C, Node > &&!std::same_as< C, Node & >) Node(C &&c)
- const NodeContent & c () const
- NodeContent & c ()

0.15.320.1 Constructor & Destructor Documentation

0.15.320.1.1 Node() `smtrat::covering_ng::formula::node_ds::Node::Node ( const Node & c ) [inline]`

0.15.320.2 Member Function Documentation

0.15.320.2.1 c() [1/2] NodeContent& smtrat::covering_ng::formula::node_ds::Node::c ( ) [inline]

0.15.320.2.2 c() [2/2] const NodeContent& smtrat::covering_ng::formula::node_ds::Node::c ( ) const [inline]

0.15.320.2.3 requires() template<typename C > smtrat::covering_ng::formula::node_ds::Node::requires (!std::same_as< C, Node > &&!std::same_as< C, Node & >) && [inline]

0.15.321 smtrat::covering_ng::formula::node_ds::NodeContent Struct Reference

#include <FormulaEvaluation.h>

Public Member Functions

- template<typename C >
  NodeContent (C &&c)

Data Fields

- std::variant< TRUE, FALSE, NOT, AND, OR, IFF, XOR, BOOL, CONSTRAINT > content
- Valuation valuation
- std::size_t max_level = 0
- std::size_t max_degree = 0
- std::size_t max_total_degree = 0
- std::size_t num_subformulas = 0
- std::size_t num_constraints = 0
- carl::Variables boolean_variables
- carl::Variables arithmetic_variables

0.15.321.1 Constructor & Destructor Documentation

0.15.321.1.1 NodeContent() template<typename C > smtrat::covering_ng::formula::node_ds::NodeContent::NodeContent (C && *c*) [inline]

0.15.321.2 Field Documentation

0.15.321.2.1 arithmetic_variables carl::Variables smtrat::covering_ng::formula::node_ds::Node↩ Content::arithmetic_variables

0.15.321.2.2 boolean_variables carl::Variables smtrat::covering_ng::formula::node_ds::Node↩ Content::boolean_variables

0.15.321.2.3 content std::variant<TRUE,FALSE,NOT,AND,OR,IFF,XOR,BOOL,CONSTRAINT> smtrat↩ ::covering_ng::formula::node_ds::NodeContent::content

0.15.321.2.4 max_degree `std::size_t smtrat::covering_ng::formula::node_ds::NodeContent::max_degree = 0`

0.15.321.2.5 max_level `std::size_t smtrat::covering_ng::formula::node_ds::NodeContent::max_level = 0`

0.15.321.2.6 max_total_degree `std::size_t smtrat::covering_ng::formula::node_ds::NodeContent::max_total_degree = 0`

0.15.321.2.7 num_constraints `std::size_t smtrat::covering_ng::formula::node_ds::NodeContent::num_constraints = 0`

0.15.321.2.8 num_subformulas `std::size_t smtrat::covering_ng::formula::node_ds::NodeContent::num_subformulas = 0`

0.15.321.2.9 valuation `Valuation smtrat::covering_ng::formula::node_ds::NodeContent::valuation`

0.15.322 smtrat::mcsat::onecellcad::recursive::NoHeuristic Struct Reference

`#include <Explanation.h>`

Static Public Attributes

- static constexpr int heuristic = 0

0.15.322.1 Field Documentation

0.15.322.1.1 heuristic `constexpr int smtrat::mcsat::onecellcad::recursive::NoHeuristic::heuristic = 0 [static], [constexpr]`

0.15.323 smtrat::covering_ng::CoveringResult< op >::NONE Struct Reference

`#include <types.h>`

0.15.324 smtrat::covering_ng::formula::node_ds::NOT Struct Reference

`#include <FormulaEvaluation.h>`

Data Fields

- Node subformula

0.15.324.1 Field Documentation

0.15.324.1.1 subformula `Node smtrat::covering_ng::formula::node_ds::NOT::subformula`

0.15.325 smtrat::NRAILModule< Settings > Class Template Reference

`#include <NRAILModule.h>`

Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- NRAILModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=nullptr)
- ~NRAILModule ()
- bool informCore (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void init ()

  Informs all backends about the so far encountered constraints, which have not yet been communicated.
- bool addCore (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- void removeCore (ModuleInput::const_iterator _subformula)

  Removes the subformula of the received formula at the given position to the considered ones of this module.
- void updateModel () const

  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 bool informCore ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.
- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)
- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.

- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

  Copies the infeasible subsets of the passed formula.
- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const

- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.

- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.325.1 Member Typedef Documentation

0.15.325.1.1 SettingsType `template<typename Settings >`

```
typedef Settings smtrat::NRAILModule< Settings >::SettingsType
```

0.15.325.2 Member Enumeration Documentation

0.15.325.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

0.15.325.3 Constructor & Destructor Documentation

0.15.325.3.1 NRAILModule() `template<typename Settings >`

```
smtrat::NRAILModule< Settings >::NRAILModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

0.15.325.3.2 ~NRAILModule() `template<typename Settings >`

```
smtrat::NRAILModule< Settings >::~NRAILModule ( )
```

0.15.325.4 Member Function Documentation

0.15.325.4.1 add() `bool smtrat::Module::add (`

```
            ModuleInput::const_iterator _subformula ) [inherited]
```

The module has to take the given sub-formula of the received formula into account.

Parameters

_subformula	The sub-formula to take additionally into account.

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

0.15.325.4.2 addConstraintToInform()

```
void smtrat::Module::addConstraintToInform (
    const FormulaT & _constraint ) [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

0.15.325.4.3 addCore() [1/2]

```
virtual bool smtrat::Module::addCore (
    [[maybe_unused] ] 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.

0.15.325.4.4 addCore() [2/2]

```
template<typename Settings >
bool smtrat::NRAILModule< Settings >::addCore (
    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.

0.15.325.4.5 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula
(
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.325.4.6 addLemma()

```
void smtrat::Module::addLemma (
    const FormulaT & _lemma,
    const LemmaType & _lt = LemmaType::NORMAL,
```

```
const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline], [inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.325.4.7 addOrigin()

```
void smtrat::Module::addOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.325.4.8 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.325.4.9 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.325.4.10 addSubformulaToPassedFormula() [2/3] std::pair<ModuleInput::iterator,bool> smtrat::Module::addSubformulaToPassedFormula (const FormulaT & _formula, const FormulaT & _origin) [inline], [protected], [inherited]

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.325.4.11 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.

0.15.325.4.12 allModels() const std::vector<Model>& smtrat::Module::allModels ( ) const [inline], [inherited]

Returns

All satisfying assignments, if existent.

0.15.325.4.13 anAnswerFound() bool smtrat::Module::anAnswerFound ( ) const [inline], [protected], [inherited]

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.325.4.14 answerFound() `const smtrat::Conditionals& smtrat::Module::answerFound ( ) const [inline], [inherited]`

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.325.4.15 backendsModel() `const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.325.4.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.325.4.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.325.4.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.325.4.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_polynomial	The variable to branch for.

Parameters

_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

0.15.325.4.20 check()

```
Answer smtrat::Module::check (
    bool _final = false,
    bool _full = true,
    carl::Variable _objective = carl::Variable::NO_VARIABLE ) [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.325.4.21 checkCore()

```
template<typename Settings >
Answer smtrat::NRAILModule< Settings >::checkCore ( ) [virtual]
```

Checks the received formula for consistency.

Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.325.4.22 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.

0.15.325.4.23 checkModel() `unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]`

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.325.4.24 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.325.4.25 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

0.15.325.4.26 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

0.15.325.4.27 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

0.15.325.4.28 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.325.4.29 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

0.15.325.4.30 collectOrigins() [2/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.
If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.325.4.31 collectTheoryPropagations()

`void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

0.15.325.4.32 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ToInform ( ) const [inline], [inherited]`

Returns

The constraints which the backends must be informed about.

0.15.325.4.33 currentlySatisfied()

`virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.325.4.34 currentlySatisfiedByBackend()

`unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.325.4.35 deinform()

`void smtrat::Module::deinform ( const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.325.4.36 deinformCore()

`virtual void smtrat::Module::deinformCore ( [[maybe_unused] ] 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.

0.15.325.4.37 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

0.15.325.4.38 eraseSubformulaFromPassedFormula()

`ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula ( ModuleInput::iterator _subformula, bool _ignoreOrigins = false ) [protected], [virtual], [inherited]`

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_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 >.

0.15.325.4.39 excludeNotReceivedVariablesFromModel()

`void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

0.15.325.4.40 findBestOrigin()

`std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin ( const std::vector< FormulaT > & _origins ) const [protected], [inherited]`

Parameters

_origins	

Returns

0.15.325.4.41 firstSubformulaToPass() `ModuleInput::const_iterator smtrat::Module::firstSubformula↩ ToPass ( ) const [inline], [inherited]`

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.325.4.42 firstUncheckedReceivedSubformula() `ModuleInput::const_iterator smtrat::Module↩ ::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.325.4.43 freeSplittingVariable() `static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.325.4.44 generateTrivialInfeasibleSubset() `void smtrat::Module::generateTrivialInfeasible↩ Subset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.325.4.45 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.325.4.46 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.325.4.47 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.325.4.48 getInfeasibleSubsets() [2/2] `std::vector< FormulaSetT > smtrat::Module::get↩ InfeasibleSubsets ( 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.

0.15.325.4.49 getInformationRelevantFormulas() `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.325.4.50 getModelEqualities() `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.325.4.51 getOrigins() [1/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.325.4.52 getOrigins() [2/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.325.4.53 getOrigins() [3/3] `const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]`

Gets the origins of the passed formula at the given position.

Parameters

_formula	The position of a formula in the passed formulas.

Returns

The origins of the passed formula at the given position.

0.15.325.4.54 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.

0.15.325.4.55 hasLemmas() `bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

0.15.325.4.56 hasValidInfeasibleSubset() `bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]`

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.325.4.57 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.325.4.58 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.325.4.59 inform() `bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.
It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

Parameters

_constraint	The constraint to inform about.

Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

0.15.325.4.60 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.

0.15.325.4.61 informCore() [1/2]

`virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.325.4.62 informCore() [2/2]

`template<typename Settings > bool smtrat::NRAILModule< Settings >::informCore ( 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.

0.15.325.4.63 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.

0.15.325.4.64 init() template<typename 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.

0.15.325.4.65 is_minimizing() bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]

0.15.325.4.66 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.

0.15.325.4.67 isPreprocessor() bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.325.4.68 lemmas() const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.325.4.69 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.

0.15.325.4.70 model() const Model& smtrat::Module::model ( ) const [inline], [inherited]

Returns

The assignment of the current satisfiable result, if existent.

0.15.325.4.71 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.

0.15.325.4.72 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.

0.15.325.4.73 objective()

```
carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]
```

0.15.325.4.74 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
    const FormulaT & _origin ) const [protected], [inherited]
```

0.15.325.4.75 passedFormulaBegin()

```
ModuleInput::iterator smtrat::Module::passedFormulaBegin (
) [inline], [protected], [inherited]
```

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.325.4.76 passedFormulaEnd()

```
ModuleInput::iterator smtrat::Module::passedFormulaEnd ( )
[inline], [protected], [inherited]
```

Returns

An iterator to the end of the passed formula.

0.15.325.4.77 pPassedFormula()

```
const ModuleInput* smtrat::Module::pPassedFormula ( ) const
[inline], [inherited]
```

Returns

A pointer to the formula passed to the backends of this module.

0.15.325.4.78 pReceivedFormula() `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.325.4.79 print() `void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.325.4.80 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.

0.15.325.4.81 printInfeasibleSubsets() `void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.325.4.82 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.

0.15.325.4.83 printPassedFormula() `void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.325.4.84 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.325.4.85 probablyLooping()

`bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.325.4.86 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.325.4.87 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

0.15.325.4.88 receivedVariable()

`bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

0.15.325.4.89 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.

0.15.325.4.90 removeCore() [1/2]

```
virtual void smtrat::Module::removeCore (
    [[maybe_unused] ] 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.

0.15.325.4.91 removeCore() [2/2]

```
template<typename Settings >
void smtrat::NRAILModule< Settings >::removeCore (
    ModuleInput::const_iterator _subformula )
```

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.

0.15.325.4.92 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

0.15.325.4.93 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

0.15.325.4.94 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.325.4.95 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to this module.

0.15.325.4.96 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( ) [inline],
[protected], [virtual], [inherited]
```

Reimplemented in smtrat::PModule.

0.15.325.4.97 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.

0.15.325.4.98 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.

0.15.325.4.99 setThreadPriority()

```
void smtrat::Module::setThreadPriority (
    thread_priority _threadPriority ) [inline], [inherited]
```

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.325.4.100 solverState()

```
Answer smtrat::Module::solverState ( ) const [inline], [inherited]
```

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.325.4.101 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 !=.

0.15.325.4.102 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.325.4.103 updateAllModels() `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.325.4.104 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.325.4.105 updateModel() `template<typename Settings > void smtrat::NRAILModule< Settings >::updateModel ( ) const [virtual]`

Updates the current assignment into the model.
Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.
Reimplemented from smtrat::Module.

0.15.325.4.106 usedBackends() `const std::vector<Module*>& smtrat::Module::usedBackends ( ) const [inline], [inherited]`

Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.325.5 Field Documentation

0.15.325.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.325.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.325.5.3 mBackendsFoundAnswer `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.325.5.4 mConstraintsToInform

`carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.325.5.5 mFinalCheck

`bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.325.5.6 mFirstPosInLastBranches

`std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.325.5.7 mFirstSubformulaToPass

`ModuleInput::iterator smtrat::Module::mFirstSubformulaTo↩ Pass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.325.5.8 mFirstUncheckedReceivedSubformula

`ModuleInput::const_iterator smtrat::Module↩ ::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.325.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.325.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.325.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.325.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.325.5.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std↩ ::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.325.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.325.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.325.5.16 mModelComputed `bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.325.5.17 mNumOfBranchVarsToStore `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.325.5.18 mObjectiveVariable `carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.325.5.19 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

0.15.325.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.325.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.325.5.22 mSolverState `std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.325.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

0.15.325.5.24 mUsedBackends `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.325.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.326 smtrat::parser::NumeralParser Struct Reference

Parses numerals: `(0 | [1-9][0-9]*)`

```
#include <Lexicon.h>
```

0.15.326.1 Detailed Description

Parses numerals: `(0 | [1-9][0-9]*)`

0.15.327 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 is_positive () const

  Checks whether this Numeric corresponds to a positive rational number.
- bool is_negative () const
- bool is_zero () const
- bool is_integer () const

0.15.327.1 Constructor & Destructor Documentation

0.15.327.1.1 Numeric() [1/8] `smtrat::lra::Numeric::Numeric ( )`

Default constructor.

0.15.327.1.2 Numeric() [2/8] `smtrat::lra::Numeric::Numeric ( const Rational & _value )`

Constructing from a Rational.

Parameters

The	Rational.

0.15.327.1.3 Numeric() [3/8] `smtrat::lra::Numeric::Numeric ( int _value )`

Constructing from an integer.

Parameters

_value	The integer.

0.15.327.1.4 Numeric() [4/8] `smtrat::lra::Numeric::Numeric ( unsigned int _value )`

Constructing from an unsigned integer.

Parameters

_value	The unsigned integer

0.15.327.1.5 Numeric() [5/8] `smtrat::lra::Numeric::Numeric ( long _value )`

Constructing from a long integer.

Parameters

_value	The unsigned long integer.

0.15.327.1.6 Numeric() [6/8] `smtrat::lra::Numeric::Numeric (`

```
unsigned long _value )
```

Constructing from an unsigned long integer.

Parameters

_value	The unsigned long integer.

0.15.327.1.7 Numeric() [7/8]

```
smtrat::lra::Numeric::Numeric (
const char * _value )
```

Constructing from a char array.

Parameters

_value	The char array.

0.15.327.1.8 Numeric() [8/8]

```
smtrat::lra::Numeric::Numeric (
const Numeric & _value )
```

Copy constructor.

Parameters

_value	The Numeric to copy.

0.15.327.1.9 ~Numeric()

```
smtrat::lra::Numeric::~Numeric ( )
```

0.15.327.2 Member Function Documentation

0.15.327.2.1 content()

```
const Rational& smtrat::lra::Numeric::content ( ) const [inline]
```

0.15.327.2.2 denom()

```
Numeric smtrat::lra::Numeric::denom ( ) const
```

Returns

The denominator of this Numeric.

0.15.327.2.3 floor()

```
Numeric smtrat::lra::Numeric::floor ( ) const
```

Returns

The next smaller integer to this Numeric.

0.15.327.2.4 is_integer()

```
bool smtrat::lra::Numeric::is_integer ( ) const
```

Returns

true, if this Numeric is integer; false, otherwise.

0.15.327.2.5 is_negative() `bool smtrat::lra::Numeric::is_negative ( ) const`

Returns

true, if this Numeric corresponds to a positive rational number; false, otherwise.

0.15.327.2.6 is_positive() `bool smtrat::lra::Numeric::is_positive ( ) const`

Checks whether this Numeric corresponds to a positive rational number.

Returns

True, if this Numeric corresponds to a positive rational number; False, otherwise.

0.15.327.2.7 is_zero() `bool smtrat::lra::Numeric::is_zero ( ) const`

Returns

true, if this Numeric corresponds to zero; false, otherwise.

0.15.327.2.8 numer() `Numeric smtrat::lra::Numeric::numer ( ) const`

Returns

The enumerator of this Numeric.

0.15.327.2.9 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.

0.15.327.2.10 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.

0.15.327.2.11 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.

0.15.327.2.12 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.

0.15.327.2.13 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.

0.15.327.2.14 operator=() [3/6] `Numeric & smtrat::lra::Numeric::operator= ( int _value )`

Cast from an integer.

Parameters

_value	The integer.

Returns

The corresponding Numeric.

0.15.327.2.15 operator=() [4/6] `Numeric & smtrat::lra::Numeric::operator= ( long _value )`

Cast from a long integer.

Parameters

_value	The long integer.

Returns

The corresponding Numeric.

0.15.327.2.16 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.

0.15.327.2.17 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.

0.15.327.2.18 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.

0.15.327.2.19 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.

0.15.327.2.20 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.

0.15.327.2.21 rContent() `Rational& smtrat::lra::Numeric::rContent ( ) [inline]`

0.15.328 smtrat::execution::Objective Struct Reference

`#include <ExecutionState.h>`

Data Fields

- Poly function
- bool minimize

0.15.328.1 Field Documentation

0.15.328.1.1 function `Poly smtrat::execution::Objective::function`

0.15.328.1.2 minimize `bool smtrat::execution::Objective::minimize`

0.15.329 Minisat::OccLists< Idx, Vec, Deleted > Class Template Reference

`#include <SolverTypes.h>`

Public Member Functions

- OccLists (const Deleted &d)
- void init (const Idx &idx)
- const Vec & operator[] (const Idx &idx) const
- Vec & operator[] (const Idx &idx)
- Vec & lookup (const Idx &idx)
- void cleanAll ()
- void clean (const Idx &idx)
- void smudge (const Idx &idx)
- void clear (bool free=true)

0.15.329.1 Constructor & Destructor Documentation

0.15.329.1.1 OccLists()

```
template<class Idx , class Vec , class Deleted >
Minisat::OccLists< Idx, Vec, Deleted >::OccLists (
            const Deleted & d )  [inline]
```

0.15.329.2 Member Function Documentation

0.15.329.2.1 clean()

```
template<class Idx , class Vec , class Deleted >
void Minisat::OccLists< Idx, Vec, Deleted >::clean (
            const Idx & idx )
```

0.15.329.2.2 cleanAll()

```
template<class Idx , class Vec , class Deleted >
void Minisat::OccLists< Idx, Vec, Deleted >::cleanAll
```

0.15.329.2.3 clear()

```
template<class Idx , class Vec , class Deleted >
void Minisat::OccLists< Idx, Vec, Deleted >::clear (
            bool free = true )  [inline]
```

0.15.329.2.4 init()

```
template<class Idx , class Vec , class Deleted >
void Minisat::OccLists< Idx, Vec, Deleted >::init (
            const Idx & idx )  [inline]
```

0.15.329.2.5 lookup()

```
template<class Idx , class Vec , class Deleted >
Vec& Minisat::OccLists< Idx, Vec, Deleted >::lookup (
            const Idx & idx )  [inline]
```

0.15.329.2.6 operator[]() [1/2]

```
template<class Idx , class Vec , class Deleted >
Vec& Minisat::OccLists< Idx, Vec, Deleted >::operator[] (
            const Idx & idx )  [inline]
```

0.15.329.2.7 operator[]() [2/2]

```
template<class Idx , class Vec , class Deleted >
const Vec& Minisat::OccLists< Idx, Vec, Deleted >::operator[] (
            const Idx & idx ) const  [inline]
```

0.15.329.2.8 smudge()

```
template<class Idx , class Vec , class Deleted >
void Minisat::OccLists< Idx, Vec, Deleted >::smudge (
            const Idx & idx )  [inline]
```

0.15.330 smtrat::mcsat::onecellcad::OneCellCAD Class Reference

```
#include <utils.h>
```

Public Member Functions

- OneCellCAD (const std::vector< carl::Variable > &variableOrder, const RealAlgebraicPoint< Rational > &point)
- std::map< carl::Variable, RAN > prefixPointToStdMap (const std::size_t componentCount)

  Create a mapping from the first variables (with index 0 to 'componentCount') in the 'variableOrder' to the first components of the given 'point'.
- std::vector< RAN > isolateLastVariableRoots (const std::size_t polyLevel, const Poly &poly)

  src/lib/datastructures/mcsat/onecellcad/OneCellCAD.h Given a poly p(x1,..,xn), return all roots of the univariate poly p(a1,..,an-1, xn), where a1,..an-1 are the first algebraic real components from 'point' (SORTED!).
- bool vanishesEarly (const std::size_t polyLevel, const Poly &poly)

  Check if an n-variate 'poly' p(x1,..,xn) with n>=1 already becomes the zero poly after plugging in p(a1,..,an-1, xn), where a1,..an-1 are the first n-1 algebraic real components from 'point'.
- bool isPointRootOfPoly (const std::size_t polyLevel, const Poly &poly)
- bool isPointRootOfPoly (const TagPoly &poly)
- std::optional< Poly > coeffNonNull (const TagPoly &boundCandidate)
- bool isMainPointInsideCell (const CADCell &cell)

Data Fields

- const std::vector< carl::Variable > & variableOrder

  Variables can also be indexed by level.
- const RealAlgebraicPoint< Rational > & point

0.15.330.1 Constructor & Destructor Documentation

0.15.330.1.1 OneCellCAD()

```
smtrat::mcsat::onecellcad::OneCellCAD::OneCellCAD (
    const std::vector< carl::Variable > & variableOrder,
    const RealAlgebraicPoint< Rational > & point ) [inline]
```

0.15.330.2 Member Function Documentation

0.15.330.2.1 coeffNonNull()

```
std::optional<Poly> smtrat::mcsat::onecellcad::OneCellCAD::coeff↩
NonNull (
    const TagPoly & boundCandidate ) [inline]
```

0.15.330.2.2 isMainPointInsideCell()

```
bool smtrat::mcsat::onecellcad::OneCellCAD::isMainPoint↩
InsideCell (
    const CADCell & cell ) [inline]
```

0.15.330.2.3 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!).

0.15.330.2.4 isPointRootOfPoly() [1/2]

```
bool smtrat::mcsat::onecellcad::OneCellCAD::isPointRootOfPoly (
    const std::size_t polyLevel,
    const Poly & poly ) [inline]
```

0.15.330.2.5 isPointRootOfPoly() [2/2]

```
bool smtrat::mcsat::onecellcad::OneCellCAD::isPointRootOfPoly (
    const TagPoly & poly ) [inline]
```

0.15.330.2.6 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'.

0.15.330.2.7 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'.

0.15.330.3 Field Documentation

0.15.330.3.1 point

```
const RealAlgebraicPoint<Rational>& smtrat::mcsat::onecellcad::OneCellCAD::point
```

0.15.330.3.2 variableOrder

```
const std::vector<carl::Variable>& smtrat::mcsat::onecellcad::OneCellCAD::variableOrder
```

Variables can also be indexed by level.
Polys with mathematical level 1 contain the variable in variableOrder[0]

0.15.331 benchmax::settings::OperationSettings Struct Reference

Operation settings.

```
#include <Settings.h>
```

Data Fields

- std::string backend

  Name of the backend to be used.
- std::string mode

  Name of the operation mode.
- std::string convert_ods_filename

  Name of the xml file to convert to an ods file.
- std::string convert_ods_filter

  Name of the xslt filter used for ods import.
- bool use_temp

  Use temporary directory.

0.15.331.1 Detailed Description

Operation settings.

0.15.331.2 Field Documentation

0.15.331.2.1 backend `std::string benchmax::settings::OperationSettings::backend`

Name of the backend to be used.

0.15.331.2.2 convert_ods_filename `std::string benchmax::settings::OperationSettings::convert_↩ods_filename`

Name of the xml file to convert to an ods file.

0.15.331.2.3 convert_ods_filter `std::string benchmax::settings::OperationSettings::convert_ods↩_filter`

Name of the xslt filter used for ods import.

0.15.331.2.4 mode `std::string benchmax::settings::OperationSettings::mode`

Name of the operation mode.

0.15.331.2.5 use_temp `bool benchmax::settings::OperationSettings::use_temp`

Use temporary directory.

0.15.332 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)

0.15.332.1 Constructor & Destructor Documentation

0.15.332.1.1 Optimization()

```
template<typename Solver >
smtrat::Optimization< Solver >::Optimization (
            Solver & s )  [inline]
```

0.15.332.2 Member Function Documentation

0.15.332.2.1 active()

```
template<typename Solver >
bool smtrat::Optimization< Solver >::active ( ) const  [inline]
```

0.15.332.2.2 add_objective()

```
template<typename Solver >
void smtrat::Optimization< Solver >::add_objective (
            const Poly & objective,
            bool minimize = true )  [inline]
```

0.15.332.2.3 compute()

```
template<typename Solver >
std::tuple<Answer,Model,ObjectiveValues> smtrat::Optimization< Solver >::compute (
            bool full = true )  [inline]
```

0.15.332.2.4 remove_objective()

```
template<typename Solver >
void smtrat::Optimization< Solver >::remove_objective (
            const Poly & objective )  [inline]
```

0.15.332.2.5 reset()

```
template<typename Solver >
void smtrat::Optimization< Solver >::reset ( )  [inline]
```

0.15.333 Minisat::Option Class Reference

```
#include <Options.h>
```

Data Structures

- struct OptionLt

Public Member Functions

- virtual ~Option ()
- virtual void help (bool verbose=false)=0

Protected Member Functions

- Option (const char *name_, const char *desc_, const char *cate_, const char *type_)

Static Protected Member Functions

- static vec< Option * > & getOptionList ()
- static const char *& getUsageString ()
- static const char *& getHelpPrefixString ()

Protected Attributes

- const char * name
- const char * description
- const char * category
- const char * type_name

Friends

- void printUsageAndExit (int argc, char **argv, bool verbose)
- void setUsageHelp (const char *str)
- void setHelpPrefixStr (const char *str)

0.15.333.1 Constructor & Destructor Documentation

0.15.333.1.1 Option()

```
Minisat::Option::Option (
    const char * name_,
    const char * desc_,
    const char * cate_,
    const char * type_ )  [inline], [protected]
```

0.15.333.1.2 ~Option()

```
virtual Minisat::Option::~Option ( )  [inline], [virtual]
```

0.15.333.2 Member Function Documentation

0.15.333.2.1 getHelpPrefixString()

```
static const char*& Minisat::Option::getHelpPrefixString ( )
[inline], [static], [protected]
```

0.15.333.2.2 getOptionList()

```
static vec<Option*>& Minisat::Option::getOptionList ( )  [inline],
[static], [protected]
```

0.15.333.2.3 getUsageString()

```
static const char*& Minisat::Option::getUsageString ( )  [inline],
[static], [protected]
```

0.15.333.2.4 help()

```
virtual void Minisat::Option::help (
    bool verbose = false )  [pure virtual]
```

Implemented in Minisat::BoolOption, Minisat::StringOption, Minisat::IntOption, and Minisat::DoubleOption.

0.15.333.3 Friends And Related Function Documentation

0.15.333.3.1 printUsageAndExit

```
void printUsageAndExit (
    int argc,
    char ** argv,
    bool verbose )  [friend]
```

0.15.333.3.2 setHelpPrefixStr

```
void setHelpPrefixStr (
    const char * str )  [friend]
```

0.15.333.3.3 setUsageHelp

```
void setUsageHelp (
    const char * str )  [friend]
```

0.15.333.4 Field Documentation

0.15.333.4.1 category

```
const char* Minisat::Option::category  [protected]
```

0.15.333.4.2 description

```
const char* Minisat::Option::description  [protected]
```

0.15.333.4.3 name `const char* Minisat::Option::name [protected]`

0.15.333.4.4 type_name `const char* Minisat::Option::type_name [protected]`

0.15.334 Minisat::Option::OptionLt Struct Reference

`#include <Options.h>`

Public Member Functions

- bool operator() (const Option *x, const Option *y)

0.15.334.1 Member Function Documentation

0.15.334.1.1 operator()()

```
bool Minisat::Option::OptionLt::operator() (
    const Option * x,
    const Option * y ) [inline]
```

0.15.335 smtrat::covering_ng::formula::node_ds::OR Struct Reference

`#include <FormulaEvaluation.h>`

Data Fields

- std::vector< Node > subformulas

0.15.335.1 Field Documentation

0.15.335.1.1 subformulas `std::vector<Node> smtrat::covering_ng::formula::node_ds::OR::subformulas`

0.15.336 smtrat::cad::Origin Class Reference

This class represents one or more origins of some object.

`#include <Origin.h>`

Data Structures

- struct BaseType

Public Member Functions

- Origin ()
- Origin (const Origin &po)
- Origin (Origin &&po)
- Origin (std::size_t level, std::size_t id)
- Origin (const BaseType &bt)
- auto begin () const
- auto end () const
- bool isActive () const
- void deactivate (std::size_t level, const carl::Bitset &rhs)
- void deactivateEC (std::size_t level, const carl::Bitset &rhs)
- void activate (std::size_t level, const carl::Bitset &rhs)
- void activateEC (std::size_t level, const carl::Bitset &rhs)

- Origin & operator= (const Origin &rhs)
- Origin & operator= (Origin &&rhs)
- bool empty () const
- bool operator== (const Origin &rhs) const
- Origin & operator+= (const BaseType &rhs)

  Adds the pair to the origins.
- Origin & operator-= (const BaseType &rhs)

  Removes the pair from the origins.
- Origin & erase (std::size_t level, const carl::Bitset &rhs)
- Origin operator| (const Origin &rhs) const

Friends

- std::ostream & operator<< (std::ostream &os, const Origin &po)

0.15.336.1 Detailed Description

This class represents one or more origins of some object.
An origin is a single id or a pair of ids (where a single id is represented as a pair of the same id).
As an object can have multiple origins, this class stores a list of such pairs. This class makes sure that the pairs are unique.

0.15.336.2 Constructor & Destructor Documentation

0.15.336.2.1 Origin() [1/5] `smtrat::cad::Origin::Origin ( ) [inline]`

0.15.336.2.2 Origin() [2/5] `smtrat::cad::Origin::Origin ( const Origin & po ) [inline]`

0.15.336.2.3 Origin() [3/5] `smtrat::cad::Origin::Origin ( Origin && po ) [inline]`

0.15.336.2.4 Origin() [4/5] `smtrat::cad::Origin::Origin ( std::size_t level, std::size_t id ) [inline], [explicit]`

0.15.336.2.5 Origin() [5/5] `smtrat::cad::Origin::Origin ( const BaseType & bt ) [inline], [explicit]`

0.15.336.3 Member Function Documentation

0.15.336.3.1 activate() `void smtrat::cad::Origin::activate ( std::size_t level, const carl::Bitset & rhs ) [inline]`

0.15.336.3.2 activateEC()

```
void smtrat::cad::Origin::activateEC (
std::size_t level,
const carl::Bitset & rhs ) [inline]
```

0.15.336.3.3 begin()

```
auto smtrat::cad::Origin::begin ( ) const [inline]
```

0.15.336.3.4 deactivate()

```
void smtrat::cad::Origin::deactivate (
std::size_t level,
const carl::Bitset & rhs ) [inline]
```

0.15.336.3.5 deactivateEC()

```
void smtrat::cad::Origin::deactivateEC (
std::size_t level,
const carl::Bitset & rhs ) [inline]
```

0.15.336.3.6 empty()

```
bool smtrat::cad::Origin::empty ( ) const [inline]
```

0.15.336.3.7 end()

```
auto smtrat::cad::Origin::end ( ) const [inline]
```

0.15.336.3.8 erase()

```
Origin& smtrat::cad::Origin::erase (
std::size_t level,
const carl::Bitset & rhs ) [inline]
```

0.15.336.3.9 isActive()

```
bool smtrat::cad::Origin::isActive ( ) const [inline]
```

0.15.336.3.10 operator+=()

```
Origin& smtrat::cad::Origin::operator+= (
const BaseType & rhs ) [inline]
```

Adds the pair to the origins.

0.15.336.3.11 operator-=()

```
Origin& smtrat::cad::Origin::operator-= (
const BaseType & rhs ) [inline]
```

Removes the pair from the origins.

0.15.336.3.12 operator=() [1/2]

```
Origin& smtrat::cad::Origin::operator= (
const Origin & rhs ) [inline]
```

0.15.336.3.13 operator=() [2/2]

```
Origin& smtrat::cad::Origin::operator= (
Origin && rhs ) [inline]
```

0.15.336.3.14 operator==()

```
bool smtrat::cad::Origin::operator== (
const Origin & rhs ) const [inline]
```

0.15.336.3.15 operator"|() `Origin smtrat::cad::Origin::operator| ( const Origin & rhs ) const [inline]`

0.15.336.4 Friends And Related Function Documentation

0.15.336.4.1 operator<< `std::ostream& operator<< ( std::ostream & os, const Origin & po ) [friend]`

0.15.337 smtrat::cad::preprocessor::Origins Struct Reference

`#include <CADPreprocessor.h>`

Public Member Functions

- void cleanOrigins (const FormulaT &f)
- void add (const FormulaT &f, const std::vector< FormulaT > &origin)
- void remove (const FormulaT &f)
- const std::vector< FormulaT > & get (const FormulaT &f) const
- bool resolve (std::set< FormulaT > &conflict) const

Data Fields

- std::map< FormulaT, std::vector< std::vector< FormulaT > > > mOrigins

0.15.337.1 Member Function Documentation

0.15.337.1.1 add() `void smtrat::cad::preprocessor::Origins::add ( const FormulaT & f, const std::vector< FormulaT > & origin )`

0.15.337.1.2 cleanOrigins() `void smtrat::cad::preprocessor::Origins::cleanOrigins ( const FormulaT & f )`

0.15.337.1.3 get() `const std::vector< FormulaT > & smtrat::cad::preprocessor::Origins::get ( const FormulaT & f ) const`

0.15.337.1.4 remove() `void smtrat::cad::preprocessor::Origins::remove ( const FormulaT & f )`

0.15.337.1.5 resolve() `bool smtrat::cad::preprocessor::Origins::resolve ( std::set< FormulaT > & conflict ) const`

0.15.337.2 Field Documentation

0.15.337.2.1 mOrigins `std::map<FormulaT, std::vector<std::vector<FormulaT> > > smtrat::cad↩ ::preprocessor::Origins::mOrigins`

0.15.338 Minisat::OutOfMemoryException Class Reference

`#include <XAlloc.h>`

0.15.339 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)

0.15.339.1 Constructor & Destructor Documentation

0.15.339.1.1 OutputWrapper() [1/2]

```
smtrat::parser::OutputWrapper::OutputWrapper (
    std::ostream & o,
    const std::string & prefix,
    const std::string & suffix ) [inline]
```

0.15.339.1.2 OutputWrapper() [2/2]

```
smtrat::parser::OutputWrapper::OutputWrapper (
    const OutputWrapper && o ) [inline]
```

0.15.339.1.3 ~OutputWrapper()

```
smtrat::parser::OutputWrapper::~OutputWrapper ( ) [inline]
```

0.15.339.2 Member Function Documentation

0.15.339.2.1 operator<<()

```
template<typename T >
OutputWrapper& smtrat::parser::OutputWrapper::operator<< (
    const T & t ) [inline]
```

0.15.340 Minisat::Map< K, D, H, E >::Pair Struct Reference

`#include <Map.h>`

Data Fields

- K key
- D data

0.15.340.1 Field Documentation

0.15.340.1.1 data

```
template<class K , class D , class H = Hash<K>, class E = Equal<K>>
D Minisat::Map< K, D, H, E >::Pair::data
```

0.15.340.1.2 key `template<class K , class D , class H = Hash<K>, class E = Equal<K>>`

`K Minisat::Map< K, D, H, E >::Pair::key`

0.15.341 smtrat::mcsat::ParallelExplanation< Backends > Struct Template Reference

`#include <ParallelExplanation.h>`

Public Member Functions

- std::optional< Explanation > operator() (const mcsat::Bookkeeping &data, carl::Variable var, const FormulasT &reason, bool force_use_core) const

0.15.341.1 Member Function Documentation

0.15.341.1.1 operator()() `template<typename... Backends>`

```
std::optional<Explanation> smtrat::mcsat::ParallelExplanation< Backends >::operator() (
            const mcsat::Bookkeeping & data,
            carl::Variable var,
            const FormulasT & reason,
            bool force_use_core ) const  [inline]
```

0.15.342 smtrat::parser::ParserSettings Struct Reference

`#include <ParserSettings.h>`

Data Fields

- bool read_dimacs
- bool read_opb
- std::string input_file
- bool disable_uf_flattening
- bool disable_theory

0.15.342.1 Field Documentation

0.15.342.1.1 disable_theory `bool smtrat::parser::ParserSettings::disable_theory`

0.15.342.1.2 disable_uf_flattening `bool smtrat::parser::ParserSettings::disable_uf_flattening`

0.15.342.1.3 input_file `std::string smtrat::parser::ParserSettings::input_file`

0.15.342.1.4 read_dimacs `bool smtrat::parser::ParserSettings::read_dimacs`

0.15.342.1.5 read_opb `bool smtrat::parser::ParserSettings::read_opb`

0.15.343 smtrat::parser::ParserState Struct Reference

`#include <ParserState.h>`

Data Structures

- struct ExpressionScope
- struct ScriptScope

Public Member Functions

- ParserState (InstructionHandler &ih)
- ~ParserState ()
- void pushExpressionScope ()
- void popExpressionScope ()
- void pushScriptScope ()
- void popScriptScope ()
- std::size_t script_scope_size () const
- void reset ()
- void errorMessage (const std::string &msg)
- bool isSymbolFree (const std::string &name, bool output=true)
- template<typename Res , typename T >
  bool resolveSymbol (const std::string &name, const std::map< std::string, T > &map, Res &result) const
- bool resolveSymbol (const std::string &name, types::TermType &r) const
- bool resolveSymbol (const std::string &name, types::VariableType &r) const
- void registerFunction (const std::string &name, const FunctionInstantiator *fi)
- void registerFunction (const std::string &name, const IndexedFunctionInstantiator *fi)
- void registerFunction (const std::string &name, const UserFunctionInstantiator *fi)

Data Fields

- carl::Logic logic
- std::set< types::VariableType > auxiliary_variables
- std::map< std::string, types::TermType > bindings
- std::map< std::string, types::TermType > constants
- std::map< std::string, types::VariableType > variables
- std::map< std::string, carl::UninterpretedFunction > declared_functions
- std::map< std::string, const FunctionInstantiator * > defined_functions
- std::map< std::string, const IndexedFunctionInstantiator * > defined_indexed_functions
- std::map< std::string, const UserFunctionInstantiator * > defined_user_functions
- FormulasT global_formulas
- std::vector< smtrat::ModelVariable > artificialVariables
- InstructionHandler & handler
- std::stack< ExpressionScope > expressionScopes
- std::stack< ScriptScope > scriptScopes

0.15.343.1 Constructor & Destructor Documentation

0.15.343.1.1 ParserState() `smtrat::parser::ParserState::ParserState ( InstructionHandler & ih ) [inline]`

0.15.343.1.2 ~ParserState() `smtrat::parser::ParserState::~ParserState ( ) [inline]`

0.15.343.2 Member Function Documentation

0.15.343.2.1 errorMessage()

```
void smtrat::parser::ParserState::errorMessage (
    const std::string & msg ) [inline]
```

0.15.343.2.2 isSymbolFree()

```
bool smtrat::parser::ParserState::isSymbolFree (
    const std::string & name,
    bool output = true ) [inline]
```

0.15.343.2.3 popExpressionScope()

```
void smtrat::parser::ParserState::popExpressionScope ( ) [inline]
```

0.15.343.2.4 popScriptScope()

```
void smtrat::parser::ParserState::popScriptScope ( ) [inline]
```

0.15.343.2.5 pushExpressionScope()

```
void smtrat::parser::ParserState::pushExpressionScope ( ) [inline]
```

0.15.343.2.6 pushScriptScope()

```
void smtrat::parser::ParserState::pushScriptScope ( ) [inline]
```

0.15.343.2.7 registerFunction() [1/3]

```
void smtrat::parser::ParserState::registerFunction (
    const std::string & name,
    const FunctionInstantiator * fi ) [inline]
```

0.15.343.2.8 registerFunction() [2/3]

```
void smtrat::parser::ParserState::registerFunction (
    const std::string & name,
    const IndexedFunctionInstantiator * fi ) [inline]
```

0.15.343.2.9 registerFunction() [3/3]

```
void smtrat::parser::ParserState::registerFunction (
    const std::string & name,
    const UserFunctionInstantiator * fi ) [inline]
```

0.15.343.2.10 reset()

```
void smtrat::parser::ParserState::reset ( ) [inline]
```

0.15.343.2.11 resolveSymbol() [1/3]

```
template<typename Res , typename T >
bool smtrat::parser::ParserState::resolveSymbol (
    const std::string & name,
    const std::map< std::string, T > & map,
    Res & result ) const [inline]
```

0.15.343.2.12 resolveSymbol() [2/3]

```
bool smtrat::parser::ParserState::resolveSymbol (
    const std::string & name,
    types::TermType & r ) const [inline]
```

0.15.343.2.13 resolveSymbol() [3/3]

```
bool smtrat::parser::ParserState::resolveSymbol (
    const std::string & name,
    types::VariableType & r ) const [inline]
```

0.15.343.2.14 script_scope_size()

```
std::size_t smtrat::parser::ParserState::script_scope_size ( ) const [inline]
```

0.15.343.3 Field Documentation

0.15.343.3.1 artificialVariables

```
std::vector<smtrat::ModelVariable> smtrat::parser::ParserState↩
::artificialVariables
```

0.15.343.3.2 auxiliary_variables

```
std::set<types::VariableType> smtrat::parser::ParserState↩
::auxiliary_variables
```

0.15.343.3.3 bindings

```
std::map<std::string, types::TermType> smtrat::parser::ParserState↩
::bindings
```

0.15.343.3.4 constants

```
std::map<std::string, types::TermType> smtrat::parser::ParserState↩
::constants
```

0.15.343.3.5 declared_functions

```
std::map<std::string, carl::UninterpretedFunction> smtrat↩
::parser::ParserState::declared_functions
```

0.15.343.3.6 defined_functions

```
std::map<std::string, const FunctionInstantiator*> smtrat↩
::parser::ParserState::defined_functions
```

0.15.343.3.7 defined_indexed_functions

```
std::map<std::string, const IndexedFunctionInstantiator*>
smtrat::parser::ParserState::defined_indexed_functions
```

0.15.343.3.8 defined_user_functions

```
std::map<std::string, const UserFunctionInstantiator*>
smtrat::parser::ParserState::defined_user_functions
```

0.15.343.3.9 expressionScopes

```
std::stack<ExpressionScope> smtrat::parser::ParserState::expression↩
Scopes
```

0.15.343.3.10 global_formulas

```
FormulasT smtrat::parser::ParserState::global_formulas
```

0.15.343.3.11 handler

```
InstructionHandler& smtrat::parser::ParserState::handler
```

0.15.343.3.12 logic `carl::Logic smtrat::parser::ParserState::logic`

0.15.343.3.13 scriptScopes `std::stack<ScriptScope> smtrat::parser::ParserState::scriptScopes`

0.15.343.3.14 variables `std::map<std::string, types::VariableType> smtrat::parser::Parser↩State::variables`

0.15.344 smtrat::PBGaussModule< Settings > Class Template Reference

`#include <PBGaussModule.h>`

Public Types

- using MatrixT = Eigen::Matrix< Rational, Eigen::Dynamic, Eigen::Dynamic >
- using VectorT = Eigen::Matrix< Rational, Eigen::Dynamic, 1 >
- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- PBGaussModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=nullptr)
- ~PBGaussModule ()
- bool informCore (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void init ()

  Informs all backends about the so far encountered constraints, which have not yet been communicated.
- bool addCore (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- void removeCore (ModuleInput::const_iterator _subformula)

  Removes the subformula of the received formula at the given position to the considered ones of this module.
- void updateModel () const

  Updates the current assignment into the model.
- Answer checkCore ()

  Checks the received formula for consistency.
- FormulaT gaussAlgorithm ()
- FormulaT reconstructEqSystem (const MatrixT &m, const VectorT &b, const carl::Variables &vars, const std↩::vector< carl::Relation > &rels)
- FormulaT reduce (const MatrixT &u, const VectorT &b, const carl::Variables vars)
- std::vector< Rational > lookForReductionRow (const MatrixT &uMatrix, const VectorT &ineqRow, long column)
- bool inform (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void deinform (const FormulaT &_constraint)

  The inverse of informing about a constraint.
- bool add (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- virtual Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_↩VARIABLE)

  Checks the received formula for consistency.
- virtual void remove (ModuleInput::const_iterator _subformula)

  Removes everything related to the given sub-formula of the received formula.

- virtual void updateAllModels ()

  Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
- virtual std::list< std::vector< carl::Variable > > getModelEqualities () const

  Partition the variables from the current model into equivalence classes according to their assigned value.
- unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const
- virtual unsigned currentlySatisfied (const FormulaT &) const
- bool receivedVariable (carl::Variable::Arg _var) const
- Answer solverState () const
- std::size_t id () const
- void setId (std::size_t _id)

  Sets this modules unique ID to identify itself.
- thread_priority threadPriority () const
- void setThreadPriority (thread_priority _threadPriority)

  Sets the priority of this module to get a thread for running its check procedure.
- const ModuleInput * pReceivedFormula () const
- const ModuleInput & rReceivedFormula () const
- const ModuleInput * pPassedFormula () const
- const ModuleInput & rPassedFormula () const
- const Model & model () const
- const std::vector< Model > & allModels () const
- const std::vector< FormulaSetT > & infeasibleSubsets () const
- const std::vector< Module * > & usedBackends () const
- const carl::FastSet< FormulaT > & constraintsToInform () const
- const carl::FastSet< FormulaT > & informedConstraints () const
- void addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))

  Stores a lemma being a valid formula.
- bool hasLemmas ()

  Checks whether this module or any of its backends provides any lemmas.
- void clearLemmas ()

  Deletes all yet found lemmas.
- const std::vector< Lemma > & lemmas () const
- ModuleInput::const_iterator firstUncheckedReceivedSubformula () const
- ModuleInput::const_iterator firstSubformulaToPass () const
- void receivedFormulaChecked ()

  Notifies that the received formulas has been checked.
- const 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 bool informCore ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.

- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)
- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

  Copies the infeasible subsets of the passed formula.
- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

Merges the two vectors of sets into the first one.

- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.344.1 Member Typedef Documentation

0.15.344.1.1 MatrixT `template<typename Settings >`

```
using smtrat::PBGaussModule< Settings >::MatrixT = Eigen::Matrix<Rational, Eigen::Dynamic,
Eigen::Dynamic>
```

0.15.344.1.2 SettingsType `template<typename Settings >`

```
typedef Settings smtrat::PBGaussModule< Settings >::SettingsType
```

0.15.344.1.3 VectorT `template<typename Settings >`

```
using smtrat::PBGaussModule< Settings >::VectorT = Eigen::Matrix<Rational, Eigen::Dynamic, 1>
```

0.15.344.2 Member Enumeration Documentation

0.15.344.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

0.15.344.3 Constructor & Destructor Documentation

0.15.344.3.1 PBGaussModule() `template<typename Settings >`

```
smtrat::PBGaussModule< Settings >::PBGaussModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

0.15.344.3.2 ~PBGaussModule() `template<typename Settings >`

```
smtrat::PBGaussModule< Settings >::~PBGaussModule ( )
```

0.15.344.4 Member Function Documentation

0.15.344.4.1 add() `bool smtrat::Module::add (`

```
            ModuleInput::const_iterator _subformula )  [inherited]
```

The module has to take the given sub-formula of the received formula into account.

Parameters

_subformula	The sub-formula to take additionally into account.

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

0.15.344.4.2 addConstraintToInform() `void smtrat::Module::addConstraintToInform (`

```
            const FormulaT & _constraint )  [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

0.15.344.4.3 addCore() `[1/2]` `virtual bool smtrat::Module::addCore (`

```
            [[maybe_unused] ] 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.

0.15.344.4.4 addCore() [2/2] template<typename Settings >

```
bool smtrat::PBGaussModule< Settings >::addCore (
            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.

0.15.344.4.5 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula
(
            const FormulaT & formula )  [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.344.4.6 addLemma()

```
void smtrat::Module::addLemma (
            const FormulaT & _lemma,
            const LemmaType & _lt = LemmaType::NORMAL,
            const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) )  [inline],
[inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.344.4.7 addOrigin()

```
void smtrat::Module::addOrigin (
            ModuleInput::iterator _formula,
            const FormulaT & _origin )  [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.344.4.8 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
            ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.344.4.9 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.344.4.10 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula,
            const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.344.4.11 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.

0.15.344.4.12 allModels() `const std::vector<Model>& smtrat::Module::allModels ( ) const [inline], [inherited]`

Returns

All satisfying assignments, if existent.

0.15.344.4.13 anAnswerFound() `bool smtrat::Module::anAnswerFound ( ) const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.344.4.14 answerFound() `const smtrat::Conditionals& smtrat::Module::answerFound ( ) const [inline], [inherited]`

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.344.4.15 backendsModel() `const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.344.4.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.344.4.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.344.4.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline],
[protected], [inherited]
```

0.15.344.4.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.344.4.20 check()

```
Answer smtrat::Module::check (
    bool _final = false,
    bool _full = true,
    carl::Variable _objective = carl::Variable::NO_VARIABLE ) [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.344.4.21 checkCore()

`template<typename 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.

0.15.344.4.22 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.

0.15.344.4.23 checkModel()

`unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]`

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.344.4.24 cleanModel()

`void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.344.4.25 clearLemmas()

`void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

0.15.344.4.26 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

0.15.344.4.27 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

0.15.344.4.28 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.344.4.29 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

0.15.344.4.30 collectOrigins() [2/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.
If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.344.4.31 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

0.15.344.4.32 conservativeResize() [1/2]

```
template<typename Settings >
static void smtrat::PBGaussModule< Settings >::conservativeResize (
    MatrixT & m,
    long newRows,
    long newCols ) [inline], [static]
```

Convenience wrapper for Eigen::conservativeResizeLike with a zero matrix.

0.15.344.4.33 conservativeResize() [2/2]

```
template<typename Settings >
static void smtrat::PBGaussModule< Settings >::conservativeResize (
    VectorT & v,
    long newSize ) [inline], [static]
```

Convenience wrapper for Eigen::conservativeResizeLike with a zero vector.

0.15.344.4.34 constraintsToInform() `const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ToInform ( ) const [inline], [inherited]`

Returns

The constraints which the backends must be informed about.

0.15.344.4.35 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

0.15.344.4.36 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)

0.15.344.4.37 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.

0.15.344.4.38 deinformCore()

`virtual void smtrat::Module::deinformCore ( [[maybe_unused] ] 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.

0.15.344.4.39 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

0.15.344.4.40 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 >.

0.15.344.4.41 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

0.15.344.4.42 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
            const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

0.15.344.4.43 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.

0.15.344.4.44 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.

0.15.344.4.45 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.344.4.46 gaussAlgorithm()

`template<typename Settings > FormulaT smtrat::PBGaussModule< Settings >::gaussAlgorithm ( )`

0.15.344.4.47 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasible↩Subset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.344.4.48 getBackendsAllModels()

`void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.344.4.49 getBackendsModel()

`void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.344.4.50 getInfeasibleSubsets() [1/2]

`void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.344.4.51 getInfeasibleSubsets() [2/2]

`std::vector< FormulaSetT > smtrat::Module::get↩InfeasibleSubsets ( 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.

0.15.344.4.52 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module↩::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.344.4.53 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.

0.15.344.4.54 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
const FormulaT & _formula,
FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.344.4.55 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
const FormulaT & _formula,
FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.344.4.56 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.

0.15.344.4.57 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.

0.15.344.4.58 hasLemmas()

```
bool smtrat::Module::hasLemmas ( ) [inline], [inherited]
```

Checks whether this module or any of its backends provides any lemmas.

0.15.344.4.59 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).

0.15.344.4.60 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.344.4.61 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.

0.15.344.4.62 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.

0.15.344.4.63 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.

0.15.344.4.64 informCore() [1/2] `virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.344.4.65 informCore() [2/2]

```
template<typename Settings >
bool smtrat::PBGaussModule< Settings >::informCore (
            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.

0.15.344.4.66 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.

0.15.344.4.67 init()

```
template<typename 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.

0.15.344.4.68 is_minimizing()

```
bool smtrat::Module::is_minimizing ( ) const  [inline], [inherited]
```

0.15.344.4.69 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.

0.15.344.4.70 isPreprocessor()

```
bool smtrat::Module::isPreprocessor ( ) const  [inline], [inherited]
```

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.344.4.71 lemmas() `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.344.4.72 lookForReductionRow() `template<typename Settings > std::vector<Rational> smtrat::PBGaussModule< Settings >::lookForReductionRow (`

```
const MatrixT & uMatrix,
const VectorT & ineqRow,
long column )
```

0.15.344.4.73 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.

0.15.344.4.74 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.344.4.75 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.

0.15.344.4.76 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.

0.15.344.4.77 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

0.15.344.4.78 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.344.4.79 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.344.4.80 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.344.4.81 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.344.4.82 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.344.4.83 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.344.4.84 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.

0.15.344.4.85 printInfeasibleSubsets()

`void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.344.4.86 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.

0.15.344.4.87 printPassedFormula()

`void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.344.4.88 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.344.4.89 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.

0.15.344.4.90 receivedFormulaChecked()

```
void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]
```

Notifies that the received formulas has been checked.

0.15.344.4.91 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩InfeasibleSubset (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

0.15.344.4.92 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const [inline], [inherited]
```

0.15.344.4.93 reconstructEqSystem()

```
template<typename Settings >
FormulaT smtrat::PBGaussModule< Settings >::reconstructEqSystem (
    const MatrixT & m,
    const VectorT & b,
    const carl::Variables & vars,
    const std::vector< carl::Relation > & rels )
```

0.15.344.4.94 reduce()

```
template<typename Settings >
FormulaT smtrat::PBGaussModule< Settings >::reduce (
    const MatrixT & u,
    const VectorT & b,
    const carl::Variables vars )
```

0.15.344.4.95 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.

0.15.344.4.96 removeCore() [1/2]

```
virtual void smtrat::Module::removeCore (
            [[maybe_unused] ] 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.

0.15.344.4.97 removeCore() [2/2]

```
template<typename Settings >
void smtrat::PBGaussModule< Settings >::removeCore (
            ModuleInput::const_iterator _subformula )
```

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.

0.15.344.4.98 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origin (
            ModuleInput::iterator _formula,
            const FormulaT & _origin ) [inline], [protected], [inherited]
```

0.15.344.4.99 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

0.15.344.4.100 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.344.4.101 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to this module.

0.15.344.4.102 runBackends() [1/2]

`virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]`

Reimplemented in smtrat::PModule.

0.15.344.4.103 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.

0.15.344.4.104 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.

0.15.344.4.105 setThreadPriority()

```
void smtrat::Module::setThreadPriority (
    thread_priority _threadPriority ) [inline], [inherited]
```

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.344.4.106 solverState()

`Answer smtrat::Module::solverState ( ) const [inline], [inherited]`

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.344.4.107 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 !=.

0.15.344.4.108 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.344.4.109 updateAllModels()

`void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.344.4.110 updateLemmas()

`void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.344.4.111 updateModel()

`template<typename Settings > void smtrat::PBGaussModule< Settings >::updateModel ( ) const [virtual]`

Updates the current assignment into the model.
Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.
Reimplemented from smtrat::Module.

0.15.344.4.112 usedBackends()

`const std::vector<Module*>& smtrat::Module::usedBackends ( ) const [inline], [inherited]`

Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.344.5 Field Documentation

0.15.344.5.1 mAllBackends

`std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.344.5.2 mAllModels

`std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.344.5.3 mBackendsFoundAnswer

`std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.344.5.4 mConstraintsToInform

`carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.344.5.5 mFinalCheck

`bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.344.5.6 mFirstPosInLastBranches

`std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.344.5.7 mFirstSubformulaToPass

`ModuleInput::iterator smtrat::Module::mFirstSubformulaToPass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.344.5.8 mFirstUncheckedReceivedSubformula

`ModuleInput::const_iterator smtrat::Module::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.344.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.344.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.344.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.344.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.344.5.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.344.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.344.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.344.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.344.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.344.5.18 mObjectiveVariable

`carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.344.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]`

Reusable splitting variables.

0.15.344.5.20 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.344.5.21 mSmallerMusesCheckCounter

`unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.344.5.22 mSolverState

`std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.344.5.23 mTheoryPropagations

`std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]`

0.15.344.5.24 mUsedBackends `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.344.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.345 smtrat::PBPPModule< Settings > Class Template Reference

`#include <PBPPModule.h>`

Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- PBPPModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=nullptr)
- ~PBPPModule ()
- bool informCore (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void init ()

  Informs all backends about the so far encountered constraints, which have not yet been communicated.
- bool addCore (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- void removeCore (ModuleInput::const_iterator _subformula)

  Removes the subformula of the received formula at the given position to the considered ones of this module.
- void updateModel () const

  Updates the current assignment into the model.
- Answer checkCore ()

  Checks the received formula for consistency.
- bool inform (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void deinform (const FormulaT &_constraint)

  The inverse of informing about a constraint.
- bool add (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- virtual Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_↩VARIABLE)

  Checks the received formula for consistency.
- virtual void remove (ModuleInput::const_iterator _subformula)

  Removes everything related to the given sub-formula of the received formula.
- virtual void updateAllModels ()

  Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
- virtual std::list< std::vector< carl::Variable > > getModelEqualities () const

  Partition the variables from the current model into equivalence classes according to their assigned value.
- unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const
- virtual unsigned currentlySatisfied (const FormulaT &) const
- bool receivedVariable (carl::Variable::Arg _var) const

- Answer solverState () const
- std::size_t id () const
- void setId (std::size_t _id)

  Sets this modules unique ID to identify itself.
- thread_priority threadPriority () const
- void setThreadPriority (thread_priority _threadPriority)

  Sets the priority of this module to get a thread for running its check procedure.
- const ModuleInput ∗ pReceivedFormula () const
- const ModuleInput & rReceivedFormula () const
- const ModuleInput ∗ pPassedFormula () const
- const ModuleInput & rPassedFormula () const
- const Model & model () const
- const std::vector< Model > & allModels () const
- const std::vector< FormulaSetT > & infeasibleSubsets () const
- const std::vector< Module ∗ > & usedBackends () const
- const carl::FastSet< FormulaT > & constraintsToInform () const
- const carl::FastSet< FormulaT > & informedConstraints () const
- void addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))

  Stores a lemma being a valid formula.
- bool hasLemmas ()

  Checks whether this module or any of its backends provides any lemmas.
- void clearLemmas ()

  Deletes all yet found lemmas.
- const std::vector< Lemma > & lemmas () const
- ModuleInput::const_iterator firstUncheckedReceivedSubformula () const
- ModuleInput::const_iterator firstSubformulaToPass () const
- void receivedFormulaChecked ()

  Notifies that the received formulas has been checked.
- const 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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.
- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_↩ origin)

- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_↩ ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

  Copies the infeasible subsets of the passed formula.
- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_↩ branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAs↩ Premise=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 >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.

- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.345.1 Member Typedef Documentation

0.15.345.1.1 SettingsType

```
template<typename Settings >
typedef Settings smtrat::PBPPModule< Settings >::SettingsType
```

0.15.345.2 Member Enumeration Documentation

0.15.345.2.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

0.15.345.3 Constructor & Destructor Documentation

0.15.345.3.1 PBPPModule()

```
template<typename Settings >
smtrat::PBPPModule< Settings >::PBPPModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

0.15.345.3.2 ~PBPPModule()

```
template<typename Settings >
smtrat::PBPPModule< Settings >::~PBPPModule ( )
```

0.15.345.4 Member Function Documentation

0.15.345.4.1 add()

```
bool smtrat::Module::add (
            ModuleInput::const_iterator _subformula ) [inherited]
```

The module has to take the given sub-formula of the received formula into account.

Parameters

_subformula	The sub-formula to take additionally into account.

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

0.15.345.4.2 addConstraintToInform()

```
void smtrat::Module::addConstraintToInform (
    const FormulaT & _constraint ) [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

0.15.345.4.3 addCore() [1/2]

```
virtual bool smtrat::Module::addCore (
    [[maybe_unused] ] 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.

0.15.345.4.4 addCore() [2/2]

```
template<typename Settings >
bool smtrat::PBPPModule< Settings >::addCore (
    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.

0.15.345.4.5 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula
(
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.345.4.6 addLemma()

```
void smtrat::Module::addLemma (
    const FormulaT & _lemma,
    const LemmaType & _lt = LemmaType::NORMAL,
    const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline], [inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.345.4.7 addOrigin()

```
void smtrat::Module::addOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.345.4.8 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::addReceivedSubformulaToPassedFormula (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.345.4.9 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.345.4.10 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula,
            const FormulaT & _origin )  [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.345.4.11 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.

0.15.345.4.12 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const  [inline],
[inherited]
```

Returns

All satisfying assignments, if existent.

0.15.345.4.13 anAnswerFound()

`bool smtrat::Module::anAnswerFound ( ) const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.345.4.14 answerFound()

`const smtrat::Conditionals& smtrat::Module::answerFound ( ) const [inline], [inherited]`

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.345.4.15 backendsModel()

`const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.345.4.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.345.4.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.345.4.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.345.4.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
```

```
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false )  [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.345.4.20 check()

```
Answer smtrat::Module::check (
bool _final = false,
bool _full = true,
carl::Variable _objective = carl::Variable::NO_VARIABLE )  [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.345.4.21 checkCore()

```
template<typename 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.

0.15.345.4.22 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.

0.15.345.4.23 checkModel()

```
unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]
```

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.345.4.24 cleanModel()

```
void smtrat::Module::cleanModel ( ) const [inline], [protected],
[inherited]
```

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.345.4.25 clearLemmas()

```
void smtrat::Module::clearLemmas ( ) [inline], [inherited]
```

Deletes all yet found lemmas.

0.15.345.4.26 clearModel()

```
void smtrat::Module::clearModel ( ) const [inline], [protected],
[inherited]
```

Clears the assignment, if any was found.

0.15.345.4.27 clearModels()

```
void smtrat::Module::clearModels ( ) const [inline], [protected],
[inherited]
```

Clears all assignments, if any was found.

0.15.345.4.28 clearPassedFormula()

```
void smtrat::Module::clearPassedFormula ( ) [protected],
[inherited]
```

0.15.345.4.29 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
const FormulaT & _formula,
FormulaSetT & _origins ) const [inherited]
```

0.15.345.4.30 collectOrigins() [2/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.
If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.345.4.31 collectTheoryPropagations()

```
void smtrat::Module::collectTheoryPropagations ( ) [inherited]
```

0.15.345.4.32 constraintsToInform()

```
const carl::FastSet<FormulaT>& smtrat::Module::constraints↩
ToInform ( ) const [inline], [inherited]
```

Returns

The constraints which the backends must be informed about.

0.15.345.4.33 currentlySatisfied()

```
virtual unsigned smtrat::Module::currentlySatisfied (
    const FormulaT & ) const [inline], [virtual], [inherited]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.345.4.34 currentlySatisfiedByBackend()

```
unsigned smtrat::Module::currentlySatisfiedByBackend (
    const FormulaT & _formula ) const [inherited]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.345.4.35 deinform()

```
void smtrat::Module::deinform (
    const FormulaT & _constraint ) [inherited]
```

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.345.4.36 deinformCore()

`virtual void smtrat::Module::deinformCore ( [[maybe_unused] ] 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.

0.15.345.4.37 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

0.15.345.4.38 eraseSubformulaFromPassedFormula()

`ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula ( ModuleInput::iterator _subformula, bool _ignoreOrigins = false ) [protected], [virtual], [inherited]`

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_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 >.

0.15.345.4.39 excludeNotReceivedVariablesFromModel()

`void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

0.15.345.4.40 findBestOrigin()

`std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin ( const std::vector< FormulaT > & _origins ) const [protected], [inherited]`

Parameters

_origins	

Returns

0.15.345.4.41 firstSubformulaToPass() `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.345.4.42 firstUncheckedReceivedSubformula() `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.345.4.43 freeSplittingVariable() `static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.345.4.44 generateTrivialInfeasibleSubset() `void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.345.4.45 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.345.4.46 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.345.4.47 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.345.4.48 getInfeasibleSubsets() [2/2] `std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets ( const Module & _backend ) const [protected], [inherited]`

Get the infeasible subsets the given backend provides.
Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

Parameters

_backend	The backend from which to obtain the infeasible subsets.

Returns

The infeasible subsets the given backend provides.

0.15.345.4.49 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.345.4.50 getModelEqualities()

`std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.345.4.51 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
const FormulaT & _formula,
FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.345.4.52 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
const FormulaT & _formula,
FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.345.4.53 getOrigins() [3/3]

```
const FormulaT& smtrat::Module::getOrigins (
ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]
```

Gets the origins of the passed formula at the given position.

Parameters

_formula	The position of a formula in the passed formulas.

Returns

The origins of the passed formula at the given position.

0.15.345.4.54 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.

0.15.345.4.55 hasLemmas() `bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

0.15.345.4.56 hasValidInfeasibleSubset() `bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]`

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.345.4.57 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.345.4.58 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.345.4.59 inform() `bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.
It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

Parameters

_constraint	The constraint to inform about.

Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

0.15.345.4.60 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.

0.15.345.4.61 informCore() [1/2]

`virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.345.4.62 informCore() [2/2]

`template<typename Settings > bool smtrat::PBPPModule< Settings >::informCore ( 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.

0.15.345.4.63 informedConstraints()

`const carl::FastSet<FormulaT>& smtrat::Module::informedConstraints ( ) const [inline], [inherited]`

Returns

The position of the first constraint of which no backend has been informed about.

0.15.345.4.64 init() `template<typename 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.

0.15.345.4.65 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.345.4.66 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.

0.15.345.4.67 isPreprocessor() `bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.345.4.68 lemmas() `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.345.4.69 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.

0.15.345.4.70 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.345.4.71 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.

0.15.345.4.72 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.

0.15.345.4.73 objective()

```
carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]
```

0.15.345.4.74 originInReceivedFormula()

```
bool smtrat::Module::originInReceivedFormula (
    const FormulaT & _origin ) const [protected], [inherited]
```

0.15.345.4.75 passedFormulaBegin()

```
ModuleInput::iterator smtrat::Module::passedFormulaBegin (
) [inline], [protected], [inherited]
```

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.345.4.76 passedFormulaEnd()

```
ModuleInput::iterator smtrat::Module::passedFormulaEnd ( )
[inline], [protected], [inherited]
```

Returns

An iterator to the end of the passed formula.

0.15.345.4.77 pPassedFormula()

```
const ModuleInput* smtrat::Module::pPassedFormula ( ) const
[inline], [inherited]
```

Returns

A pointer to the formula passed to the backends of this module.

0.15.345.4.78 pReceivedFormula() `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.345.4.79 print() `void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.345.4.80 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.

0.15.345.4.81 printInfeasibleSubsets() `void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.345.4.82 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.

0.15.345.4.83 printPassedFormula() `void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.345.4.84 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.345.4.85 probablyLooping()

`bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.345.4.86 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.345.4.87 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

0.15.345.4.88 receivedVariable()

`bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

0.15.345.4.89 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.

0.15.345.4.90 removeCore() [1/2]

```
virtual void smtrat::Module::removeCore (
    [[maybe_unused] ] 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.

0.15.345.4.91 removeCore() [2/2]

```
template<typename Settings >
void smtrat::PBPPModule< Settings >::removeCore (
    ModuleInput::const_iterator _subformula )
```

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.

0.15.345.4.92 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

0.15.345.4.93 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

0.15.345.4.94 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.345.4.95 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to this module.

0.15.345.4.96 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( ) [inline],
[protected], [virtual], [inherited]
```

Reimplemented in smtrat::PModule.

0.15.345.4.97 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.

0.15.345.4.98 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.

0.15.345.4.99 setThreadPriority()

```
void smtrat::Module::setThreadPriority (
    thread_priority _threadPriority ) [inline], [inherited]
```

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.345.4.100 solverState()

```
Answer smtrat::Module::solverState ( ) const [inline], [inherited]
```

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.345.4.101 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 !=.

0.15.345.4.102 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.345.4.103 updateAllModels() `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.345.4.104 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.345.4.105 updateModel() `template<typename Settings > void smtrat::PBPPModule< Settings >::updateModel ( ) const [virtual]`

Updates the current assignment into the model.
Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.
Reimplemented from smtrat::Module.

0.15.345.4.106 usedBackends() `const std::vector<Module*>& smtrat::Module::usedBackends ( ) const [inline], [inherited]`

Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.345.5 Field Documentation

0.15.345.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.345.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.345.5.3 mBackendsFoundAnswer `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.345.5.4 mConstraintsToInform

`carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.345.5.5 mFinalCheck

`bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.345.5.6 mFirstPosInLastBranches

`std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.345.5.7 mFirstSubformulaToPass

`ModuleInput::iterator smtrat::Module::mFirstSubformulaTo↩ Pass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.345.5.8 mFirstUncheckedReceivedSubformula

`ModuleInput::const_iterator smtrat::Module↩ ::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.345.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.345.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.345.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.345.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.345.5.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std↩ ::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.345.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.345.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.345.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.345.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.345.5.18 mObjectiveVariable

`carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.345.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]`

Reusable splitting variables.

0.15.345.5.20 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.345.5.21 mSmallerMusesCheckCounter

`unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.345.5.22 mSolverState

`std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.345.5.23 mTheoryPropagations

`std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]`

0.15.345.5.24 mUsedBackends

`std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.345.5.25 mVariableCounters

`std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.346 smtrat::PersistentUnionFind Struct Reference

`#include <UnionFind.h>`

Public Types

- using Value = size_t
- using Representative = Value
- using Parents = immer::vector< Value >
- using Ranks = immer::vector< size_t >
- using FindState = std::pair< Parents, Representative >

Public Member Functions

- void update_parents (Parents const &parents) noexcept
- void update_parents (Parents &&parents) noexcept
- void update_ranks (Ranks const &ranks) noexcept
- void update_ranks (Ranks &&ranks) noexcept
- template<typename Ps , typename Rs >
  void update (Ps &&parents, Rs &&ranks) noexcept
- void resize (size_t size) noexcept
- void init_size (size_t size) noexcept
- FindState find_impl (Parents parents, Value const &i) const noexcept
- Representative find (Value const &val) noexcept
- PersistentUnionFind merge (Value const &a, Value const &b) noexcept
- void dump () const noexcept

Data Fields

- Parents _parents
- Ranks _ranks

0.15.346.1 Member Typedef Documentation

0.15.346.1.1 FindState `using smtrat::PersistentUnionFind::FindState = std::pair<Parents, Representative>`

0.15.346.1.2 Parents `using smtrat::PersistentUnionFind::Parents = immer::vector<Value>`

0.15.346.1.3 Ranks `using smtrat::PersistentUnionFind::Ranks = immer::vector<size_t>`

0.15.346.1.4 Representative `using smtrat::PersistentUnionFind::Representative = Value`

0.15.346.1.5 Value `using smtrat::PersistentUnionFind::Value = size_t`

0.15.346.2 Member Function Documentation

0.15.346.2.1 dump() `void smtrat::PersistentUnionFind::dump ( ) const [inline], [noexcept]`

0.15.346.2.2 find() `Representative smtrat::PersistentUnionFind::find ( Value const & val ) [inline], [noexcept]`

0.15.346.2.3 find_impl()

```
FindState smtrat::PersistentUnionFind::find_impl (
    Parents parents,
    Value const & i ) const [inline], [noexcept]
```

0.15.346.2.4 init_size()

```
void smtrat::PersistentUnionFind::init_size (
    size_t size ) [inline], [noexcept]
```

0.15.346.2.5 merge()

```
PersistentUnionFind smtrat::PersistentUnionFind::merge (
    Value const & a,
    Value const & b ) [inline], [noexcept]
```

0.15.346.2.6 resize()

```
void smtrat::PersistentUnionFind::resize (
    size_t size ) [inline], [noexcept]
```

0.15.346.2.7 update()

```
template<typename Ps , typename Rs >
void smtrat::PersistentUnionFind::update (
    Ps && parents,
    Rs && ranks ) [inline], [noexcept]
```

0.15.346.2.8 update_parents() [1/2]

```
void smtrat::PersistentUnionFind::update_parents (
    Parents && parents ) [inline], [noexcept]
```

0.15.346.2.9 update_parents() [2/2]

```
void smtrat::PersistentUnionFind::update_parents (
    Parents const & parents ) [inline], [noexcept]
```

0.15.346.2.10 update_ranks() [1/2]

```
void smtrat::PersistentUnionFind::update_ranks (
    Ranks && ranks ) [inline], [noexcept]
```

0.15.346.2.11 update_ranks() [2/2]

```
void smtrat::PersistentUnionFind::update_ranks (
    Ranks const & ranks ) [inline], [noexcept]
```

0.15.346.3 Field Documentation

0.15.346.3.1 _parents

```
Parents smtrat::PersistentUnionFind::_parents
```

0.15.346.3.2 _ranks

```
Ranks smtrat::PersistentUnionFind::_ranks
```

0.15.347 smtrat::PFEModule< Settings > Class Template Reference

```
#include <PFEModule.h>
```

Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- PFEModule (const ModuleInput ∗_formula, Conditionals &_conditionals, Manager ∗_manager=NULL)
- ∼PFEModule ()
- bool addCore (ModuleInput::const_iterator)
- Answer checkCore ()

  Checks the received formula for consistency.
- void removeCore (ModuleInput::const_iterator)
- 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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.
- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)
- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.

- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

  Copies the infeasible subsets of the passed formula.
- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.347.1 Member Typedef Documentation

0.15.347.1.1 SettingsType

```
template<typename Settings >
typedef Settings smtrat::PFEModule< Settings >::SettingsType
```

0.15.347.2 Member Enumeration Documentation

0.15.347.2.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

0.15.347.3 Constructor & Destructor Documentation

0.15.347.3.1 PFEModule()

```
template<typename Settings >
smtrat::PFEModule< Settings >::PFEModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = NULL )
```

0.15.347.3.2 ~PFEModule()

```
template<typename Settings >
smtrat::PFEModule< Settings >::~PFEModule ( )
```

0.15.347.4 Member Function Documentation

0.15.347.4.1 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.

0.15.347.4.2 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 >.

0.15.347.4.3 addCore() [1/2]

```
virtual bool smtrat::Module::addCore (
    [[maybe_unused] ] 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.

0.15.347.4.4 addCore() [2/2]

```
template<typename Settings >
bool smtrat::PFEModule< Settings >::addCore (
    ModuleInput::const_iterator )
```

0.15.347.4.5 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula
(
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.347.4.6 addLemma()

```
void smtrat::Module::addLemma (
    const FormulaT & _lemma,
    const LemmaType & _lt = LemmaType::NORMAL,
    const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline],
[inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.347.4.7 addOrigin()

```
void smtrat::Module::addOrigin (
    ModuleInput::iterator _formula,
```

```
const FormulaT & _origin )  [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.347.4.8 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
            ModuleInput::const_iterator _subformula )  [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.347.4.9 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula )  [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.347.4.10 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula,
            const FormulaT & _origin )  [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.347.4.11 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.

0.15.347.4.12 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns

All satisfying assignments, if existent.

0.15.347.4.13 anAnswerFound()

```
bool smtrat::Module::anAnswerFound ( ) const [inline], [protected],
[inherited]
```

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.347.4.14 answerFound()

```
const smtrat::Conditionals& smtrat::Module::answerFound ( ) const
[inline], [inherited]
```

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.347.4.15 appliedPreprocessing()

```
bool smtrat::PModule::appliedPreprocessing ( ) const [inline],
[inherited]
```

Returns

true, if the current received formula has been simplified and the result of this simplification is stored in the passed formula.

0.15.347.4.16 backendsModel()

```
const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]
```

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.347.4.17 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.347.4.18 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.347.4.19 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.347.4.20 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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.

Parameters

_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

0.15.347.4.21 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.

0.15.347.4.22 checkCore()

```
template<typename Settings >
Answer smtrat::PFEModule< Settings >::checkCore ( ) [virtual]
```

Checks the received formula for consistency.

Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.347.4.23 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.

0.15.347.4.24 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.

0.15.347.4.25 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.347.4.26 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

0.15.347.4.27 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

0.15.347.4.28 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

0.15.347.4.29 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.347.4.30 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

0.15.347.4.31 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.

0.15.347.4.32 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

0.15.347.4.33 constraintsToInform() `const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns

The constraints which the backends must be informed about.

0.15.347.4.34 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

0.15.347.4.35 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)

0.15.347.4.36 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.

0.15.347.4.37 deinformCore() `virtual void smtrat::Module::deinformCore ( [[maybe_unused] ] 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.

0.15.347.4.38 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

0.15.347.4.39 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 >.

0.15.347.4.40 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

0.15.347.4.41 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
            const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

0.15.347.4.42 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.

0.15.347.4.43 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.

0.15.347.4.44 freeSplittingVariable() `static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.347.4.45 generateTrivialInfeasibleSubset() `void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.347.4.46 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.347.4.47 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.347.4.48 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.347.4.49 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.

0.15.347.4.50 getInformationRelevantFormulas() `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.347.4.51 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.

0.15.347.4.52 getOrigins() [1/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.347.4.53 getOrigins() [2/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.347.4.54 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.

0.15.347.4.55 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.

0.15.347.4.56 hasLemmas() `bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

0.15.347.4.57 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).

0.15.347.4.58 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.347.4.59 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasible↩ Subsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.347.4.60 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.

0.15.347.4.61 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.

0.15.347.4.62 informCore() `virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.347.4.63 informedConstraints() `const carl::FastSet<FormulaT>& smtrat::Module::informedConstraints ( ) const [inline], [inherited]`

Returns

The position of the first constraint of which no backend has been informed about.

0.15.347.4.64 init() `void smtrat::Module::init ( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not be called twice and only before the first runBackends call.
Reimplemented in smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntBlastModule< Settings >, smtrat::FPPModule< Settings >, smtrat::CurryModule< Settings >, and smtrat::BVModule< Settings >.

0.15.347.4.65 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.347.4.66 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.

0.15.347.4.67 isPreprocessor() `bool smtrat::PModule::isPreprocessor ( ) const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.347.4.68 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.

0.15.347.4.69 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.

0.15.347.4.70 model()

```
const Model& smtrat::Module::model ( ) const [inline], [inherited]
```

Returns

The assignment of the current satisfiable result, if existent.

0.15.347.4.71 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.

0.15.347.4.72 moduleName()

```
virtual std::string smtrat::Module::moduleName ( ) const [inline],
[virtual], [inherited]
```

Returns

The name of the given module type as name.

Reimplemented in smtrat::VSModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::STropModule< Settings >, smtrat::SplitSOSModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntEqModule< Settings >, smtrat::IntBlastModule< Settings >, smtrat::IncWidthModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >,

smtrat::ICPModule< ICPSettings1 >, smtrat::GBModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::CurryModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::BVModule< Settings >, and smtrat::BEModule< Settings >.

0.15.347.4.73 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

0.15.347.4.74 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.347.4.75 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.347.4.76 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.347.4.77 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.347.4.78 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.347.4.79 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.347.4.80 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.

0.15.347.4.81 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.347.4.82 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.

0.15.347.4.83 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.347.4.84 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.347.4.85 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.

0.15.347.4.86 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.347.4.87 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

0.15.347.4.88 receivedVariable()

`bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

0.15.347.4.89 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.

0.15.347.4.90 removeCore() [1/2]

`virtual void smtrat::Module::removeCore ( [[maybe_unused] ] 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.

0.15.347.4.91 removeCore() [2/2]

`template<typename Settings > void smtrat::PFEModule< Settings >::removeCore ( ModuleInput::const_iterator )`

0.15.347.4.92 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin (
            ModuleInput::iterator _formula,
            const FormulaT & _origin ) [inline], [protected], [inherited]
```

0.15.347.4.93 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]
```

0.15.347.4.94 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.347.4.95 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]
```

Returns

A reference to the formula passed to this module.

0.15.347.4.96 runBackends() [1/2]

```
virtual Answer smtrat::PModule::runBackends ( ) [inline], [virtual], [inherited]
```

Reimplemented from smtrat::Module.

0.15.347.4.97 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.

0.15.347.4.98 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.

0.15.347.4.99 setThreadPriority()

`void smtrat::Module::setThreadPriority ( thread_priority _threadPriority ) [inline], [inherited]`

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.347.4.100 solverState()

`Answer smtrat::Module::solverState ( ) const [inline], [inherited]`

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.347.4.101 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 !=.

0.15.347.4.102 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.347.4.103 updateAllModels()

`void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.347.4.104 updateLemmas()

`void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.347.4.105 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.

0.15.347.4.106 usedBackends() `const std::vector<Module*>& smtrat::Module::usedBackends ( ) const [inline], [inherited]`

Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.347.5 Field Documentation

0.15.347.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.347.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.347.5.3 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.

0.15.347.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.347.5.5 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

0.15.347.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.347.5.7 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.

0.15.347.5.8 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.

0.15.347.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.347.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.347.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.347.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.347.5.13 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.

0.15.347.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.347.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.347.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.347.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.347.5.18 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.

0.15.347.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplitting↩Variables [static], [inherited]`

Reusable splitting variables.

0.15.347.5.20 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.347.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.347.5.22 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.

0.15.347.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

0.15.347.5.24 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).

0.15.347.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.348 smtrat::PModule Class Reference

`#include <PModule.h>`

Public Types

- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- PModule (const ModuleInput *_formula, Conditionals &_foundAnswer, Manager *_manager=nullptr, std↩ ::string module_name="PModule")
- bool isPreprocessor () const
- bool appliedPreprocessing () const
- bool add (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- void remove (ModuleInput::const_iterator _subformula)

  Removes everything related to the given sub-formula of the received formula.
- Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)

  Checks the received formula for consistency.
- Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- std::pair< bool, FormulaT > getReceivedFormulaSimplified ()
- void updateModel () const

  Updates the current assignment into the model.
- bool inform (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void deinform (const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual void init ()

Informs all backends about the so far encountered constraints, which have not yet been communicated.

- virtual void updateAllModels ()

  Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
- virtual std::list< std::vector< carl::Variable > > getModelEqualities () const

  Partition the variables from the current model into equivalence classes according to their assigned value.
- unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const
- virtual unsigned currentlySatisfied (const FormulaT &) const
- bool receivedVariable (carl::Variable::Arg _var) const
- Answer solverState () const
- std::size_t id () const
- void setId (std::size_t _id)

  Sets this modules unique ID to identify itself.
- thread_priority threadPriority () const
- void setThreadPriority (thread_priority _threadPriority)

  Sets the priority of this module to get a thread for running its check procedure.
- const ModuleInput * pReceivedFormula () const
- const ModuleInput & rReceivedFormula () const
- const ModuleInput * pPassedFormula () const
- const ModuleInput & rPassedFormula () const
- const Model & model () const
- const std::vector< Model > & allModels () const
- const std::vector< FormulaSetT > & infeasibleSubsets () const
- const std::vector< Module * > & usedBackends () const
- const carl::FastSet< FormulaT > & constraintsToInform () const
- const carl::FastSet< FormulaT > & informedConstraints () const
- void addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))

  Stores a lemma being a valid formula.
- bool hasLemmas ()

  Checks whether this module or any of its backends provides any lemmas.
- void clearLemmas ()

  Deletes all yet found lemmas.
- const std::vector< Lemma > & lemmas () const
- ModuleInput::const_iterator firstUncheckedReceivedSubformula () const
- ModuleInput::const_iterator firstSubformulaToPass () const
- void receivedFormulaChecked ()

  Notifies that the received formulas has been checked.
- const 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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] 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 ([[maybe_unused]] 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 _useReceivedFormulaAs↩Premise=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 >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.348.1 Member Enumeration Documentation

0.15.348.1.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

0.15.348.2 Constructor & Destructor Documentation

0.15.348.2.1 PModule()

```
smtrat::PModule::PModule (
    const ModuleInput * _formula,
    Conditionals & _foundAnswer,
    Manager * _manager = nullptr,
    std::string module_name = "PModule" )
```

0.15.348.3 Member Function Documentation

0.15.348.3.1 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.

0.15.348.3.2 addConstraintToInform()

```
void smtrat::Module::addConstraintToInform (
    const FormulaT & _constraint ) [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

0.15.348.3.3 addCore()

```
virtual bool smtrat::Module::addCore (
    [[maybe_unused] ] 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.

0.15.348.3.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula
(
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.348.3.5 addLemma()

```
void smtrat::Module::addLemma (
    const FormulaT & _lemma,
    const LemmaType & _lt = LemmaType::NORMAL,
    const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline],
[inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.348.3.6 addOrigin()

```
void smtrat::Module::addOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.348.3.7 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.348.3.8 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.348.3.9 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.

0.15.348.3.10 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.

0.15.348.3.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns

All satisfying assignments, if existent.

0.15.348.3.12 anAnswerFound()

```
bool smtrat::Module::anAnswerFound ( ) const [inline], [protected],
[inherited]
```

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.348.3.13 answerFound()

```
const smtrat::Conditionals& smtrat::Module::answerFound ( ) const
[inline], [inherited]
```

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.348.3.14 appliedPreprocessing()

```
bool smtrat::PModule::appliedPreprocessing ( ) const [inline]
```

Returns

true, if the current received formula has been simplified and the result of this simplification is stored in the passed formula.

0.15.348.3.15 backendsModel()

```
const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]
```

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.348.3.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.348.3.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.348.3.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.348.3.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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.

Parameters

_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

0.15.348.3.20 check()

```
Answer smtrat::PModule::check (
    bool _final = false,
    bool _full = true,
    carl::Variable _objective = carl::Variable::NO_VARIABLE ) [virtual]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.348.3.21 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::VSModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::SymmetryModule< Settings >, smtrat::STropModule< Settings >, smtrat::SplitSOSModule< Settings >, smtrat::SATModule< Settings >, smtrat::PFEModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Se smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::MCBModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASetti smtrat::LRAModule< LRASettings1 >, smtrat::IntEqModule< Settings >, smtrat::IntBlastModule< Settings >, smtrat::IncWidthModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::ICEModule< Settings >, smtrat::GBPPModule< Settings >, smtrat::GBModule< Settings >, smtrat::FPPModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::ESModule< Settings smtrat::EMModule< Settings >, smtrat::CurryModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::CoveringNGModule< Settings >, smtrat::CNFerModule, smtrat::BVModule< Settings >, and smtrat::BEModule< Settings >.

0.15.348.3.22 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.

0.15.348.3.23 checkModel() `unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]`

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.348.3.24 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.348.3.25 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

0.15.348.3.26 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

0.15.348.3.27 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

0.15.348.3.28 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.348.3.29 collectOrigins() [1/2] `void smtrat::Module::collectOrigins (`

```
const FormulaT & _formula,
FormulaSetT & _origins ) const [inherited]
```

0.15.348.3.30 collectOrigins() [2/2] `void smtrat::Module::collectOrigins (`

```
const FormulaT & _formula,
FormulasT & _origins ) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.348.3.31 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

0.15.348.3.32 constraintsToInform() `const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns

The constraints which the backends must be informed about.

0.15.348.3.33 currentlySatisfied() `virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.348.3.34 currentlySatisfiedByBackend() `unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.348.3.35 deinform() `void smtrat::Module::deinform ( const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.348.3.36 deinformCore() `virtual void smtrat::Module::deinformCore (`

[[maybe_unused]] 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.

0.15.348.3.37 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

0.15.348.3.38 eraseSubformulaFromPassedFormula()

ModuleInput::iterator smtrat::Module::erase↩SubformulaFromPassedFormula (ModuleInput::iterator *_subformula*, bool *_ignoreOrigins* = *false*) [protected], [virtual], [inherited]

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_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 >.

0.15.348.3.39 excludeNotReceivedVariablesFromModel()

void smtrat::Module::excludeNotReceived↩VariablesFromModel ( ) const [inherited]

Excludes all variables from the current model, which do not occur in the received formula.

0.15.348.3.40 findBestOrigin()

std::vector< FormulaT >::const_iterator smtrat::Module::find↩BestOrigin (const std::vector< FormulaT > & *_origins*) const [protected], [inherited]

Parameters

_origins	

Returns

0.15.348.3.41 firstSubformulaToPass() `ModuleInput::const_iterator smtrat::Module::firstSubformula↩ ToPass ( ) const [inline], [inherited]`

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.348.3.42 firstUncheckedReceivedSubformula() `ModuleInput::const_iterator smtrat::Module↩ ::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.348.3.43 freeSplittingVariable() `static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.348.3.44 generateTrivialInfeasibleSubset() `void smtrat::Module::generateTrivialInfeasible↩ Subset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.348.3.45 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.348.3.46 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.348.3.47 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.348.3.48 getInfeasibleSubsets() [2/2] `std::vector< FormulaSetT > smtrat::Module::get↩ InfeasibleSubsets ( 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.

0.15.348.3.49 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.348.3.50 getModelEqualities()

`std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.348.3.51 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.348.3.52 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.348.3.53 getOrigins() [3/3]

```
const FormulaT& smtrat::Module::getOrigins (
    ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]
```

Gets the origins of the passed formula at the given position.

Parameters

_formula	The position of a formula in the passed formulas.

Returns

The origins of the passed formula at the given position.

0.15.348.3.54 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.

0.15.348.3.55 hasLemmas() `bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

0.15.348.3.56 hasValidInfeasibleSubset() `bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]`

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.348.3.57 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.348.3.58 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.348.3.59 inform() `bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.
It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

Parameters

_constraint	The constraint to inform about.

Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

0.15.348.3.60 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.

0.15.348.3.61 informCore()

`virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.348.3.62 informedConstraints()

`const carl::FastSet<FormulaT>& smtrat::Module::informedConstraints ( ) const [inline], [inherited]`

Returns

The position of the first constraint of which no backend has been informed about.

0.15.348.3.63 init()

`void smtrat::Module::init ( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not be called twice and only before the first runBackends call.
Reimplemented in smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntBlastModule< Settings >, smtrat::FPPModule< Settings >, smtrat::CurryModule< Settings >, and smtrat::BVModule< Settings >.

0.15.348.3.64 is_minimizing()

`bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.348.3.65 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.

0.15.348.3.66 isPreprocessor() `bool smtrat::PModule::isPreprocessor ( ) const [inline]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.348.3.67 lemmas() `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.348.3.68 merge() `std::vector< FormulaT > smtrat::Module::merge ( const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.
({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

Parameters

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns

The vector being the two given vectors merged.

0.15.348.3.69 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.348.3.70 modelsDisjoint() `bool smtrat::Module::modelsDisjoint ( const Model & _modelA, const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

Parameters

_modelA	The first model to check for.
_modelB	The second model to check for.

Returns

true, if there is no variable assigned by both models; false, otherwise.

0.15.348.3.71 moduleName() `virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]`

Returns

The name of the given module type as name.

Reimplemented in smtrat::VSModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::STropModule< Settings >, smtrat::SplitSOSModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntEqModule< Settings >, smtrat::IntBlastModule< Settings >, smtrat::IncWidthModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::GBModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::CurryModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::BVModule< Settings >, and smtrat::BEModule< Settings >.

0.15.348.3.72 objective() `carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

0.15.348.3.73 originInReceivedFormula() `bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.348.3.74 passedFormulaBegin() `ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.348.3.75 passedFormulaEnd() `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.348.3.76 pPassedFormula() `const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.348.3.77 pReceivedFormula() `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.348.3.78 print() `void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.348.3.79 printAllModels() `void smtrat::Module::printAllModels ( std::ostream & _out = std::cout ) [inherited]`

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.348.3.80 printInfeasibleSubsets() `void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.348.3.81 printModel() `void smtrat::Module::printModel ( std::ostream & _out = std::cout ) const [inherited]`

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.348.3.82 printPassedFormula() `void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.348.3.83 printReceivedFormula() `void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.348.3.84 probablyLooping()

`bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.348.3.85 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.348.3.86 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

0.15.348.3.87 receivedVariable()

`bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

0.15.348.3.88 remove()

`void smtrat::PModule::remove ( ModuleInput::const_iterator _subformula ) [virtual]`

Removes everything related to the given sub-formula of the received formula.
However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

Parameters

_subformula	The sub formula of the received formula to remove.

Reimplemented from smtrat::Module.

0.15.348.3.89 removeCore()

`virtual void smtrat::Module::removeCore ( [[maybe_unused] ] 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.

0.15.348.3.90 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin (
            ModuleInput::iterator _formula,
            const FormulaT & _origin ) [inline], [protected], [inherited]
```

0.15.348.3.91 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]
```

0.15.348.3.92 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.348.3.93 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]
```

Returns

A reference to the formula passed to this module.

0.15.348.3.94 runBackends() [1/2]

```
virtual Answer smtrat::PModule::runBackends ( ) [inline], [virtual]
```

Reimplemented from smtrat::Module.

0.15.348.3.95 runBackends() [2/2]

```
Answer smtrat::PModule::runBackends (
            bool _final,
            bool _full,
            carl::Variable _objective ) [virtual]
```

Runs the backend solvers on the passed formula.

Parameters

_final	true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT.
_full	false, if this module should avoid too expensive procedures and rather return unknown instead.
_objective	if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good.

Returns

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.348.3.96 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.

0.15.348.3.97 setThreadPriority()

`void smtrat::Module::setThreadPriority ( thread_priority _threadPriority ) [inline], [inherited]`

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.348.3.98 solverState()

`Answer smtrat::Module::solverState ( ) const [inline], [inherited]`

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.348.3.99 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 !=.

0.15.348.3.100 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.348.3.101 updateAllModels() `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.348.3.102 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.348.3.103 updateModel() `void smtrat::PModule::updateModel ( ) const [virtual]`

Updates the current assignment into the model.
Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.
Reimplemented from smtrat::Module.

0.15.348.3.104 usedBackends() `const std::vector<Module*>& smtrat::Module::usedBackends ( ) const [inline], [inherited]`

Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.348.4 Field Documentation

0.15.348.4.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.348.4.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.348.4.3 mBackendsFoundAnswer `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.348.4.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.348.4.5 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

0.15.348.4.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.348.4.7 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.

0.15.348.4.8 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.

0.15.348.4.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.348.4.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.348.4.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.348.4.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.348.4.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std↩::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.348.4.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.348.4.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.348.4.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.348.4.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.348.4.18 mObjectiveVariable `carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.348.4.19 mOldSplittingVariables `std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

0.15.348.4.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.348.4.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.348.4.22 mSolverState `std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.348.4.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

0.15.348.4.24 mUsedBackends `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.348.4.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.349 smtrat::cadcells::operators::properties::poly_additional_root_outside Struct Reference

`#include <properties.h>`

Public Member Functions

- size_t level () const
- std::size_t hash_on_level () const

Data Fields

- datastructures::PolyRef poly

Static Public Attributes

- static constexpr bool is_flag = false

0.15.349.1 Member Function Documentation

0.15.349.1.1 hash_on_level() `std::size_t smtrat::cadcells::operators::properties::poly_additional_root_outside::hash_on_level ( ) const [inline]`

0.15.349.1.2 level() `size_t smtrat::cadcells::operators::properties::poly_additional_root_outside::level ( ) const [inline]`

0.15.349.2 Field Documentation

0.15.349.2.1 is_flag `constexpr bool smtrat::cadcells::operators::properties::poly_additional_root_outside::is_flag = false [static], [constexpr]`

0.15.349.2.2 poly `datastructures::PolyRef smtrat::cadcells::operators::properties::poly_additional_root_outside::poly`

0.15.350 smtrat::cadcells::operators::properties::poly_irreducible_semi_sgn_inv Struct Reference

`#include <properties.h>`

Public Member Functions

- size_t level () const
- std::size_t hash_on_level () const

Data Fields

- datastructures::PolyRef poly

Static Public Attributes

- static constexpr bool is_flag = false

0.15.350.1 Member Function Documentation

0.15.350.1.1 hash_on_level() `std::size_t smtrat::cadcells::operators::properties::poly_irreducible_semi_sgn_inv::hash_on_level ( ) const [inline]`

0.15.350.1.2 level() `size_t smtrat::cadcells::operators::properties::poly_irreducible_semi_sgn_inv::level ( ) const [inline]`

0.15.350.2 Field Documentation

0.15.350.2.1 is_flag `constexpr bool smtrat::cadcells::operators::properties::poly_irreducible_semi_sgn_inv::is_flag = false [static], [constexpr]`

0.15.350.2.2 poly `datastructures::PolyRef smtrat::cadcells::operators::properties::poly_irreducible_semi_sgn_inv::poly`

0.15.351 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

0.15.351.1 Member Function Documentation

0.15.351.1.1 hash_on_level() `std::size_t smtrat::cadcells::operators::properties::poly_irreducible_sgn_inv::hash_on_level ( ) const [inline]`

0.15.351.1.2 level() `size_t smtrat::cadcells::operators::properties::poly_irreducible_sgn_inv::level ( ) const [inline]`

0.15.351.2 Field Documentation

0.15.351.2.1 is_flag `constexpr bool smtrat::cadcells::operators::properties::poly_irreducible_sgn_inv::is_flag = false [static], [constexpr]`

0.15.351.2.2 poly `datastructures::PolyRef smtrat::cadcells::operators::properties::poly_irreducible_sgn_inv::poly`

0.15.352 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

0.15.352.1 Member Function Documentation

0.15.352.1.1 hash_on_level() `std::size_t smtrat::cadcells::operators::properties::poly_ord_inv::hash_on_level ( ) const [inline]`

0.15.352.1.2 level() `size_t smtrat::cadcells::operators::properties::poly_ord_inv::level ( ) const [inline]`

0.15.352.2 Field Documentation

0.15.352.2.1 is_flag `constexpr bool smtrat::cadcells::operators::properties::poly_ord_inv::is_flag = false [static], [constexpr]`

0.15.352.2.2 poly `datastructures::PolyRef smtrat::cadcells::operators::properties::poly_ord_inv::poly`

0.15.353 smtrat::cadcells::operators::properties::poly_ord_inv_base Struct Reference

`#include <properties.h>`

Public Member Functions

- size_t level () const
- std::size_t hash_on_level () const

Data Fields

- datastructures::PolyRef poly

Static Public Attributes

- static constexpr bool is_flag = false

0.15.353.1 Member Function Documentation

0.15.353.1.1 hash_on_level() `std::size_t smtrat::cadcells::operators::properties::poly_ord_inv_base::hash_on_level ( ) const [inline]`

0.15.353.1.2 level() `size_t smtrat::cadcells::operators::properties::poly_ord_inv_base::level ( ) const [inline]`

0.15.353.2 Field Documentation

0.15.353.2.1 is_flag `constexpr bool smtrat::cadcells::operators::properties::poly_ord_inv_base::is_flag = false [static], [constexpr]`

0.15.353.2.2 poly `datastructures::PolyRef smtrat::cadcells::operators::properties::poly_ord_↩ inv_base::poly`

0.15.354 smtrat::cadcells::operators::properties::poly_pdel Struct Reference

`#include <properties.h>`

Public Member Functions

- size_t level () const
- std::size_t hash_on_level () const

Data Fields

- datastructures::PolyRef poly

Static Public Attributes

- static constexpr bool is_flag = false

0.15.354.1 Member Function Documentation

0.15.354.1.1 hash_on_level() `std::size_t smtrat::cadcells::operators::properties::poly_pdel↩ ::hash_on_level ( ) const [inline]`

0.15.354.1.2 level() `size_t smtrat::cadcells::operators::properties::poly_pdel::level ( ) const [inline]`

0.15.354.2 Field Documentation

0.15.354.2.1 is_flag `constexpr bool smtrat::cadcells::operators::properties::poly_pdel::is_flag = false [static], [constexpr]`

0.15.354.2.2 poly `datastructures::PolyRef smtrat::cadcells::operators::properties::poly_pdel↩ ::poly`

0.15.355 smtrat::cadcells::operators::properties::poly_semi_sgn_inv Struct Reference

`#include <properties.h>`

Public Member Functions

- size_t level () const
- std::size_t hash_on_level () const

Data Fields

- datastructures::PolyRef poly

Static Public Attributes

- static constexpr bool is_flag = false

0.15.355.1 Member Function Documentation

0.15.355.1.1 hash_on_level() `std::size_t smtrat::cadcells::operators::properties::poly_semi_sgn_inv::hash_on_level ( ) const [inline]`

0.15.355.1.2 level() `size_t smtrat::cadcells::operators::properties::poly_semi_sgn_inv::level ( ) const [inline]`

0.15.355.2 Field Documentation

0.15.355.2.1 is_flag `constexpr bool smtrat::cadcells::operators::properties::poly_semi_sgn_inv::is_flag = false [static], [constexpr]`

0.15.355.2.2 poly `datastructures::PolyRef smtrat::cadcells::operators::properties::poly_semi_sgn_inv::poly`

0.15.356 smtrat::cadcells::operators::properties::poly_sgn_inv Struct Reference

`#include <properties.h>`

Public Member Functions

- size_t level () const
- std::size_t hash_on_level () const

Data Fields

- datastructures::PolyRef poly

Static Public Attributes

- static constexpr bool is_flag = false

0.15.356.1 Member Function Documentation

0.15.356.1.1 hash_on_level() `std::size_t smtrat::cadcells::operators::properties::poly_sgn_inv::hash_on_level ( ) const [inline]`

0.15.356.1.2 level() `size_t smtrat::cadcells::operators::properties::poly_sgn_inv::level ( ) const [inline]`

0.15.356.2 Field Documentation

0.15.356.2.1 is_flag `constexpr bool smtrat::cadcells::operators::properties::poly_sgn_inv::is_flag = false [static], [constexpr]`

0.15.356.2.2 poly `datastructures::PolyRef smtrat::cadcells::operators::properties::poly_sgn_↩ inv::poly`

0.15.357 smtrat::cadcells::representation::util::PolyDelineation Struct Reference

`#include <util.h>`

Data Fields

- boost::container::flat_set< std::size_t > delineated_roots
- std::size_t critical_lower_root = 0
- std::size_t critical_upper_root = 0

0.15.357.1 Field Documentation

0.15.357.1.1 critical_lower_root `std::size_t smtrat::cadcells::representation::util::PolyDelineation↩ ::critical_lower_root = 0`

0.15.357.1.2 critical_upper_root `std::size_t smtrat::cadcells::representation::util::Poly↩ Delineation::critical_upper_root = 0`

0.15.357.1.3 delineated_roots `boost::container::flat_set<std::size_t> smtrat::cadcells::representation↩ ::util::PolyDelineation::delineated_roots`

0.15.358 smtrat::cadcells::representation::util::PolyDelineations Struct Reference

`#include <util.h>`

Public Member Functions

- auto & get (const datastructures::PolyRef &poly)

Data Fields

- boost::container::flat_map< datastructures::PolyRef, PolyDelineation > data

0.15.358.1 Member Function Documentation

0.15.358.1.1 get() `auto& smtrat::cadcells::representation::util::PolyDelineations::get ( const datastructures::PolyRef & poly ) [inline]`

0.15.358.2 Field Documentation

0.15.358.2.1 data `boost::container::flat_map<datastructures::PolyRef,PolyDelineation> smtrat↩ ::cadcells::representation::util::PolyDelineations::data`

0.15.359 smtrat::cad::ProjectionPolynomialInformation::PolyInfo Struct Reference

`#include <ProjectionInformation.h>`

Data Fields

- Origin origin

  Origins of this polynomial.

0.15.359.1 Field Documentation

0.15.359.1.1 origin `Origin smtrat::cad::ProjectionPolynomialInformation::PolyInfo::origin`

Origins of this polynomial.

0.15.360 smtrat::cad::PolynomialComparator< PolynomialGetter > Struct Template Reference

`#include <PolynomialLiftingQueue.h>`

Public Member Functions

- PolynomialComparator (const PolynomialGetter *pg, std::size_t level)
- bool operator() (std::size_t lhs, std::size_t rhs) const

0.15.360.1 Constructor & Destructor Documentation

0.15.360.1.1 PolynomialComparator()

```
template<typename PolynomialGetter >
smtrat::cad::PolynomialComparator< PolynomialGetter >::PolynomialComparator (
            const PolynomialGetter * pg,
            std::size_t level ) [inline]
```

0.15.360.2 Member Function Documentation

0.15.360.2.1 operator()()

```
template<typename PolynomialGetter >
bool smtrat::cad::PolynomialComparator< PolynomialGetter >::operator() (
            std::size_t lhs,
            std::size_t rhs ) const [inline]
```

0.15.361 smtrat::cad::PolynomialLiftingQueue< PolynomialGetter > Class Template Reference

`#include <PolynomialLiftingQueue.h>`

Public Member Functions

- PolynomialLiftingQueue (const PolynomialGetter *pg, std::size_t level)
- auto insert (std::size_t id)
- auto erase (std::size_t id)
- auto begin () const
- auto end () const
- auto size () const
- void disable (std::size_t id)
- void restore (std::size_t id)

Friends

- template<typename PG >
  std::ostream & operator<< (std::ostream &os, const PolynomialLiftingQueue< PG > &plq)

0.15.361.1 Constructor & Destructor Documentation

0.15.361.1.1 PolynomialLiftingQueue()

```
template<typename PolynomialGetter >
smtrat::cad::PolynomialLiftingQueue< PolynomialGetter >::PolynomialLiftingQueue (
            const PolynomialGetter * pg,
            std::size_t level )  [inline]
```

0.15.361.2 Member Function Documentation

0.15.361.2.1 begin()

```
template<typename PolynomialGetter >
auto smtrat::cad::PolynomialLiftingQueue< PolynomialGetter >::begin ( ) const  [inline]
```

0.15.361.2.2 disable()

```
template<typename PolynomialGetter >
void smtrat::cad::PolynomialLiftingQueue< PolynomialGetter >::disable (
            std::size_t id )  [inline]
```

0.15.361.2.3 end()

```
template<typename PolynomialGetter >
auto smtrat::cad::PolynomialLiftingQueue< PolynomialGetter >::end ( ) const  [inline]
```

0.15.361.2.4 erase()

```
template<typename PolynomialGetter >
auto smtrat::cad::PolynomialLiftingQueue< PolynomialGetter >::erase (
            std::size_t id )  [inline]
```

0.15.361.2.5 insert()

```
template<typename PolynomialGetter >
auto smtrat::cad::PolynomialLiftingQueue< PolynomialGetter >::insert (
            std::size_t id )  [inline]
```

0.15.361.2.6 restore()

```
template<typename PolynomialGetter >
void smtrat::cad::PolynomialLiftingQueue< PolynomialGetter >::restore (
            std::size_t id )  [inline]
```

0.15.361.2.7 size()

```
template<typename PolynomialGetter >
auto smtrat::cad::PolynomialLiftingQueue< PolynomialGetter >::size ( ) const  [inline]
```

0.15.361.3 Friends And Related Function Documentation

0.15.361.3.1 operator<<

```
template<typename PolynomialGetter >
template<typename PG >
std::ostream& operator<< (
            std::ostream & os,
            const PolynomialLiftingQueue< PG > & plq )  [friend]
```

0.15.362 smtrat::cadcells::datastructures::PolyPool Class Reference

A pool for polynomials.
`#include <polynomials.h>`

Public Member Functions

- PolyPool (const Polynomial::ContextType &context)

  Constructs a polynomial pool with respect to a variable ordering.
- const VariableOrdering & var_order () const
- PolyRef insert (const Polynomial &poly)
- PolyRef operator() (const Polynomial &poly)
- const Polynomial & get (const PolyRef &ref) const
- const Polynomial & operator() (const PolyRef &ref) const
- bool known (const Polynomial &poly) const
- void clear_levels (size_t level)
- const Polynomial::ContextType & get_context () const

Friends

- std::ostream & operator<< (std::ostream &os, const PolyPool &data)

0.15.362.1 Detailed Description

A pool for polynomials.
The polynomials are stored in a table, that is, a list of lists of polynomials of a given level.

0.15.362.2 Constructor & Destructor Documentation

0.15.362.2.1 PolyPool() `smtrat::cadcells::datastructures::PolyPool::PolyPool ( const Polynomial::ContextType & context ) [inline]`

Constructs a polynomial pool with respect to a variable ordering.

Parameters

var_order	The variable ordering determining polynomial levels.

0.15.362.3 Member Function Documentation

0.15.362.3.1 clear_levels() `void smtrat::cadcells::datastructures::PolyPool::clear_levels ( size_t level ) [inline]`

0.15.362.3.2 get() `const Polynomial& smtrat::cadcells::datastructures::PolyPool::get ( const PolyRef & ref ) const [inline]`

0.15.362.3.3 get_context() `const Polynomial::ContextType& smtrat::cadcells::datastructures::↩ PolyPool::get_context ( ) const [inline]`

0.15.362.3.4 insert()

```
PolyRef smtrat::cadcells::datastructures::PolyPool::insert (
    const Polynomial & poly ) [inline]
```

0.15.362.3.5 known()

```
bool smtrat::cadcells::datastructures::PolyPool::known (
    const Polynomial & poly ) const [inline]
```

0.15.362.3.6 operator()() [1/2]

```
PolyRef smtrat::cadcells::datastructures::PolyPool::operator() (
    const Polynomial & poly ) [inline]
```

0.15.362.3.7 operator()() [2/2]

```
const Polynomial& smtrat::cadcells::datastructures::PolyPool↩
::operator() (
    const PolyRef & ref ) const [inline]
```

0.15.362.3.8 var_order()

```
const VariableOrdering& smtrat::cadcells::datastructures::PolyPool↩
::var_order ( ) const [inline]
```

0.15.362.4 Friends And Related Function Documentation

0.15.362.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const PolyPool & data ) [friend]
```

0.15.363 smtrat::cadcells::datastructures::detail::PolyProperties Struct Reference

```
#include <projections.h>
```

Data Fields

- std::map< PolyRef, PolyRef > res
- std::optional< PolyRef > disc
- std::optional< PolyRef > ldcf
- std::vector< PolyRef > factors_nonconst

0.15.363.1 Field Documentation

0.15.363.1.1 disc

```
std::optional<PolyRef> smtrat::cadcells::datastructures::detail::Poly↩
Properties::disc
```

0.15.363.1.2 factors_nonconst

```
std::vector<PolyRef> smtrat::cadcells::datastructures::detail↩
::PolyProperties::factors_nonconst
```

0.15.363.1.3 ldcf

```
std::optional<PolyRef> smtrat::cadcells::datastructures::detail::Poly↩
Properties::ldcf
```

0.15.363.1.4 res

```
std::map<PolyRef, PolyRef> smtrat::cadcells::datastructures::detail::Poly↩
Properties::res
```

0.15.364 smtrat::cadcells::datastructures::PolyRef Struct Reference

Refers to a polynomial.

`#include <polynomials.h>`

Data Fields

- size_t level

  The level of the polynomial.
- size_t id

  The id of the polynomial with respect to its level.

0.15.364.1 Detailed Description

Refers to a polynomial.

0.15.364.2 Field Documentation

0.15.364.2.1 id `size_t smtrat::cadcells::datastructures::PolyRef::id`

The id of the polynomial with respect to its level.

0.15.364.2.2 level `size_t smtrat::cadcells::datastructures::PolyRef::level`

The level of the polynomial.

0.15.365 smtrat::PolyTree Class Reference

`#include <PolyTree.h>`

Public Types

- enum class Type : unsigned { VARIABLE , CONSTANT , SUM , PRODUCT }

Public Member Functions

- PolyTree (const Poly &_poly)
- const PolyTree & left () const
- const PolyTree & right () const
- carl::Variable::Arg variable () const
- const Integer & constant () const
- Type type () const
- const Poly & poly () const

0.15.365.1 Member Enumeration Documentation

0.15.365.1.1 Type `enum smtrat::PolyTree::Type : unsigned [strong]`

Enumerator

VARIABLE	
CONSTANT	
SUM	
PRODUCT	

0.15.365.2 Constructor & Destructor Documentation

0.15.365.2.1 PolyTree()

```
smtrat::PolyTree::PolyTree (
    const Poly & _poly )
```

0.15.365.3 Member Function Documentation

0.15.365.3.1 constant()

```
const Integer & smtrat::PolyTree::constant ( ) const
```

0.15.365.3.2 left()

```
const PolyTree & smtrat::PolyTree::left ( ) const
```

0.15.365.3.3 poly()

```
const Poly & smtrat::PolyTree::poly ( ) const
```

0.15.365.3.4 right()

```
const PolyTree & smtrat::PolyTree::right ( ) const
```

0.15.365.3.5 type()

```
PolyTree::Type smtrat::PolyTree::type ( ) const
```

0.15.365.3.6 variable()

```
carl::Variable::Arg smtrat::PolyTree::variable ( ) const
```

0.15.366 smtrat::PolyTreeContent Class Reference

```
#include <PolyTree.h>
```

Public Member Functions

- PolyTreeContent (const Poly &_poly, PolyTree::Type _type, const PolyTree &_left, const PolyTree &_right)
- PolyTreeContent (carl::Variable::Arg _variable)
- PolyTreeContent (Integer _constant)
- ~PolyTreeContent ()
- const Poly & poly () const
- bool operator== (const PolyTreeContent &_other) const

Friends

- class PolyTree

0.15.366.1 Constructor & Destructor Documentation

0.15.366.1.1 PolyTreeContent() [1/3]

```
smtrat::PolyTreeContent::PolyTreeContent (
    const Poly & _poly,
    PolyTree::Type _type,
    const PolyTree & _left,
    const PolyTree & _right ) [inline]
```

0.15.366.1.2 PolyTreeContent() [2/3] smtrat::PolyTreeContent::PolyTreeContent (carl::Variable::Arg _variable) [inline]

0.15.366.1.3 PolyTreeContent() [3/3] smtrat::PolyTreeContent::PolyTreeContent (Integer _constant) [inline]

0.15.366.1.4 ~PolyTreeContent() smtrat::PolyTreeContent::~PolyTreeContent ( ) [inline]

0.15.366.2 Member Function Documentation

0.15.366.2.1 operator==() bool smtrat::PolyTreeContent::operator== (const PolyTreeContent & _other) const [inline]

0.15.366.2.2 poly() const Poly& smtrat::PolyTreeContent::poly ( ) const [inline]

0.15.366.3 Friends And Related Function Documentation

0.15.366.3.1 PolyTree friend class PolyTree [friend]

0.15.366.4 Field Documentation

0.15.366.4.1 mConstant Integer smtrat::PolyTreeContent::mConstant

0.15.366.4.2 mVariable carl::Variable smtrat::PolyTreeContent::mVariable

0.15.367 smtrat::PolyTreePool Class Reference

#include <PolyTreePool.h>

Public Member Functions

- const PolyTreeContent * get (const Poly &_pol)

Protected Member Functions

- PolyTreePool ()

  Constructor of the pool.
- ~PolyTreePool ()

0.15.367.1 Constructor & Destructor Documentation

0.15.367.1.1 PolyTreePool() smtrat::PolyTreePool::PolyTreePool ( ) [inline], [protected]

Constructor of the pool.

0.15.367.1.2 ~PolyTreePool() `smtrat::PolyTreePool::~PolyTreePool ( ) [inline], [protected]`

0.15.367.2 Member Function Documentation

0.15.367.2.1 get() `const PolyTreeContent * smtrat::PolyTreePool::get ( const Poly & _pol )`

0.15.368 smtrat::cad::Preprocessor Class Reference

`#include <Preprocessor.h>`

Public Member Functions

- Preprocessor (const std::vector< carl::Variable > &vars)
- void clear ()
- void addConstraint (const ConstraintT &c)
- void removeConstraint (const ConstraintT &c)
- bool preprocess ()
- void postprocessConflict (std::set< FormulaT > &mis) const
- const Model & model () const
- const auto & getConflict () const
- FormulaT simplify (const FormulaT &f) const
- template<typename Map >
  preprocessor::ConstraintUpdate result (const Map &oldC) const

Friends

- std::ostream & operator<< (std::ostream &os, const Preprocessor &cadpp)

0.15.368.1 Constructor & Destructor Documentation

0.15.368.1.1 Preprocessor() `smtrat::cad::Preprocessor::Preprocessor ( const std::vector< carl::Variable > & vars ) [inline]`

0.15.368.2 Member Function Documentation

0.15.368.2.1 addConstraint() `void smtrat::cad::Preprocessor::addConstraint ( const ConstraintT & c )`

0.15.368.2.2 clear() `void smtrat::cad::Preprocessor::clear ( ) [inline]`

0.15.368.2.3 getConflict() `const auto& smtrat::cad::Preprocessor::getConflict ( ) const [inline]`

0.15.368.2.4 model() `const Model& smtrat::cad::Preprocessor::model ( ) const [inline]`

0.15.368.2.5 postprocessConflict()

`void smtrat::cad::Preprocessor::postprocessConflict ( std::set< FormulaT > & mis ) const`

0.15.368.2.6 preprocess()

`bool smtrat::cad::Preprocessor::preprocess ( )`

0.15.368.2.7 removeConstraint()

`void smtrat::cad::Preprocessor::removeConstraint ( const ConstraintT & c )`

0.15.368.2.8 result()

`template<typename Map > preprocessor::ConstraintUpdate smtrat::cad::Preprocessor::result ( const Map & oldC ) const [inline]`

0.15.368.2.9 simplify()

`FormulaT smtrat::cad::Preprocessor::simplify ( const FormulaT & f ) const [inline]`

0.15.368.3 Friends And Related Function Documentation

0.15.368.3.1 operator<<

`std::ostream& operator<< ( std::ostream & os, const Preprocessor & cadpp ) [friend]`

0.15.369 smtrat::cad::PreprocessorSettings Struct Reference

`#include <Preprocessor.h>`

Static Public Member Functions

- static void register_settings (SettingsParser &parser)
- static bool register_hook ()

Data Fields

- bool disable_variable_elimination = false
- bool disable_resultants = false

Static Public Attributes

- static const bool dummy

0.15.369.1 Member Function Documentation

0.15.369.1.1 register_hook()

`static bool smtrat::cad::PreprocessorSettings::register_hook ( ) [inline], [static]`

0.15.369.1.2 register_settings()

`static void smtrat::cad::PreprocessorSettings::register_settings ( SettingsParser & parser ) [inline], [static]`

0.15.369.2 Field Documentation

0.15.369.2.1 disable_resultants `bool smtrat::cad::PreprocessorSettings::disable_resultants = false`

0.15.369.2.2 disable_variable_elimination `bool smtrat::cad::PreprocessorSettings::disable_variable_elimination = false`

0.15.369.2.3 dummy `const bool smtrat::cad::PreprocessorSettings::dummy [static]`

0.15.370 benchmax::settings::PresetSettings Struct Reference

`#include <PresetSettings.h>`

Data Fields

- bool rwth_slurm

  Use slurm on the RWTH HPC cluster.
- std::string rwth_slurm_name

  Name for this job.

0.15.370.1 Field Documentation

0.15.370.1.1 rwth_slurm `bool benchmax::settings::PresetSettings::rwth_slurm`

Use slurm on the RWTH HPC cluster.

0.15.370.1.2 rwth_slurm_name `std::string benchmax::settings::PresetSettings::rwth_slurm_name`

Name for this job.

0.15.371 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.

0.15.371.1 Constructor & Destructor Documentation

0.15.371.1.1 PriorityQueue() [1/2]

```
template<typename T , typename Compare = std::less<T>>
smtrat::PriorityQueue< T, Compare >::PriorityQueue ( ) [inline], [explicit]
```

0.15.371.1.2 PriorityQueue() [2/2]

```
template<typename T , typename Compare = std::less<T>>
smtrat::PriorityQueue< T, Compare >::PriorityQueue (
            const Compare & comp ) [inline], [explicit]
```

0.15.371.2 Member Function Documentation

0.15.371.2.1 begin() [1/2]

```
template<typename T , typename Compare = std::less<T>>
auto smtrat::PriorityQueue< T, Compare >::begin ( ) [inline]
```

0.15.371.2.2 begin() [2/2]

```
template<typename T , typename Compare = std::less<T>>
auto smtrat::PriorityQueue< T, Compare >::begin ( ) const [inline]
```

0.15.371.2.3 clear()

```
template<typename T , typename Compare = std::less<T>>
void smtrat::PriorityQueue< T, Compare >::clear ( ) [inline]
```

0.15.371.2.4 data() [1/2]

```
template<typename T , typename Compare = std::less<T>>
auto& smtrat::PriorityQueue< T, Compare >::data ( ) [inline]
```

0.15.371.2.5 data() [2/2]

```
template<typename T , typename Compare = std::less<T>>
const auto& smtrat::PriorityQueue< T, Compare >::data ( ) const [inline]
```

0.15.371.2.6 end() [1/2]

```
template<typename T , typename Compare = std::less<T>>
auto smtrat::PriorityQueue< T, Compare >::end ( ) [inline]
```

0.15.371.2.7 end() [2/2]

```
template<typename T , typename Compare = std::less<T>>
auto smtrat::PriorityQueue< T, Compare >::end ( ) const [inline]
```

0.15.371.2.8 erase() [1/2]

```
template<typename T , typename Compare = std::less<T>>
auto smtrat::PriorityQueue< T, Compare >::erase (
            typename std::vector< T >::const_iterator it ) [inline]
```

0.15.371.2.9 erase() [2/2]

```
template<typename T , typename Compare = std::less<T>>
auto smtrat::PriorityQueue< T, Compare >::erase (
            typename std::vector< T >::const_iterator it,
            typename std::vector< T >::const_iterator end )  [inline]
```

0.15.371.2.10 find()

```
template<typename T , typename Compare = std::less<T>>
auto smtrat::PriorityQueue< T, Compare >::find (
            const T & t ) const  [inline]
```

0.15.371.2.11 fix()

```
template<typename T , typename Compare = std::less<T>>
void smtrat::PriorityQueue< T, Compare >::fix ( )  [inline]
```

0.15.371.2.12 removeIf()

```
template<typename T , typename Compare = std::less<T>>
template<typename F >
void smtrat::PriorityQueue< T, Compare >::removeIf (
            F && f )  [inline]
```

0.15.371.3 Field Documentation

0.15.371.3.1 elements

```
T std::priority_queue< T >::elements  [inherited]
```

STL member.

0.15.372 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.

0.15.372.1 Member Typedef Documentation

0.15.372.1.1 Constraints

```
template<typename Settings >
using smtrat::cad::BaseProjection< Settings >::Constraints = CADConstraints<Settings::backtracking>
[protected], [inherited]
```

0.15.372.2 Member Function Documentation

0.15.372.2.1 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.

0.15.372.2.2 addEqConstraint() [2/2] template<typename Settings >

```
virtual carl::Bitset smtrat::cad::BaseProjection< Settings >::addEqConstraint (
            const UPoly & p,
            std::size_t cid,
            bool isBound ) [inline], [virtual], [inherited]
```

Adds the given polynomial of an equational constraint to the projection.
Reimplemented in smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >.

0.15.372.2.3 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.

0.15.372.2.4 addPolynomial() [2/2] template<typename Settings >

```
virtual carl::Bitset smtrat::cad::BaseProjection< Settings >::addPolynomial (
            const UPoly & p,
            std::size_t cid,
            bool isBound ) [pure virtual], [inherited]
```

Adds the given polynomial to the projection.
Implemented in smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >, smtrat::cad::Projection< In
smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Increme
smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, and smtrat::cad::Projection< Incrementality::SIMPLE

0.15.372.2.5 callRemoveCallback() template<typename Settings >

```
void smtrat::cad::BaseProjection< Settings >::callRemoveCallback (
            std::size_t level,
            const SampleLiftedWith & slw ) const [inline], [protected], [inherited]
```

0.15.372.2.6 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.

0.15.372.2.7 dim() template<typename Settings >

```
std::size_t smtrat::cad::BaseProjection< Settings >::dim ( ) const [inline], [inherited]
```

Returns the dimension of the projection.

0.15.372.2.8 empty() [1/2] template<typename Settings >

```
virtual bool smtrat::cad::BaseProjection< Settings >::empty ( ) [inline], [virtual], [inherited]
```

0.15.372.2.9 empty() [2/2] template<typename Settings >

```
virtual bool smtrat::cad::BaseProjection< Settings >::empty (
            std::size_t level ) const [pure virtual], [inherited]
```

Implemented in smtrat::qe::cad::Projection< Settings >, smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED
smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::ModelBasedProjection< Increme
smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, and smtrat::cad::Projection< Incrementality::FULL, Backtracking::HID

0.15.372.2.10 exportAsDot() template<typename Settings >

```
virtual void smtrat::cad::BaseProjection< Settings >::exportAsDot (
            std::ostream & ) const [inline], [virtual], [inherited]
```

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, and smtrat::cad::Projection< Incrementality::FULL,

0.15.372.2.11 freeID() template<typename Settings >

```
void smtrat::cad::BaseProjection< Settings >::freeID (
            std::size_t level,
            std::size_t id ) [inline], [protected], [inherited]
```

Frees a currently used polynomial id for the given level.

0.15.372.2.12 getID() template<typename Settings >

```
std::size_t smtrat::cad::BaseProjection< Settings >::getID (
            std::size_t level ) [inline], [protected], [inherited]
```

Returns a fresh polynomial id for the given level.

0.15.372.2.13 getOrigin() template<typename Settings >

```
virtual Origin smtrat::cad::BaseProjection< Settings >::getOrigin (
            std::size_t level,
            std::size_t id ) const [inline], [virtual], [inherited]
```

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, BT, Settings >.

0.15.372.2.14 getPolyForLifting() template<typename Settings >

```
OptionalID smtrat::cad::BaseProjection< Settings >::getPolyForLifting (
            std::size_t level,
            SampleLiftedWith & slw ) [inline], [inherited]
```

Get a polynomial from this level suited for lifting.

0.15.372.2.15 getPolynomialById() template<typename Settings >

```
virtual const UPoly& smtrat::cad::BaseProjection< Settings >::getPolynomialById (
            std::size_t level,
            std::size_t id ) const [pure virtual], [inherited]
```

Retrieves a polynomial from its id.
Implemented in smtrat::qe::cad::Projection< Settings >, smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED
smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::ModelBasedProjection< Increme
smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, and smtrat::cad::Projection< Incrementality::FULL, Backtracking::HID

0.15.372.2.16 hasPolynomialById() template<typename Settings >

```
virtual bool smtrat::cad::BaseProjection< Settings >::hasPolynomialById (
            std::size_t level,
            std::size_t id ) const [pure virtual], [inherited]
```

Implemented in smtrat::qe::cad::Projection< Settings >, smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED
smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::ModelBasedProjection< Increme
smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, and smtrat::cad::Projection< Incrementality::FULL, Backtracking::HID

0.15.372.2.17 isPurged()

```
template<typename Settings >
bool smtrat::cad::BaseProjection< Settings >::isPurged (
            std::size_t level,
            std::size_t id )  [inline], [protected], [inherited]
```

0.15.372.2.18 removePolynomial() [1/2]

```
template<typename Settings >
void smtrat::cad::BaseProjection< Settings >::removePolynomial (
            const Poly & p,
            std::size_t cid,
            bool isBound )  [inline], [inherited]
```

Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.

0.15.372.2.19 removePolynomial() [2/2]

```
template<typename Settings >
virtual void smtrat::cad::BaseProjection< Settings >::removePolynomial (
            const UPoly & p,
            std::size_t cid,
            bool isBound )  [pure virtual], [inherited]
```

Removes the given polynomial from the projection.
Implemented in smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::ModelBasedProje
smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, S
smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >, and smtrat::cad::Projection< Incrementality::NONE, Backtracking::

0.15.372.2.20 reset()

```
template<typename Settings >
void smtrat::cad::BaseProjection< Settings >::reset ( )  [inline], [inherited]
```

Resets all datastructures, use the given variables from now on.

0.15.372.2.21 setRemoveCallback()

```
template<typename Settings >
template<typename F >
void smtrat::cad::BaseProjection< Settings >::setRemoveCallback (
            F && f )  [inline], [inherited]
```

Sets a callback that is called whenever polynomials are removed.

0.15.372.2.22 size() [1/2]

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::size ( ) const  [inline], [inherited]
```

0.15.372.2.23 size() [2/2]

```
template<typename Settings >
virtual std::size_t smtrat::cad::BaseProjection< Settings >::size (
            std::size_t level ) const  [pure virtual], [inherited]
```

Implemented in smtrat::qe::cad::Projection< Settings >, smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED
smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::ModelBasedProjection< Increme
smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, and smtrat::cad::Projection< Incrementality::FULL, Backtracking::HID

0.15.372.2.24 var() template<typename Settings >

carl::Variable smtrat::cad::BaseProjection< Settings >::var (std::size_t *level*) const [inline], [protected], [inherited]

Returns the variable that corresponds to the given level, that is the variable eliminated in this level.

0.15.372.2.25 vars() template<typename Settings >

const auto& smtrat::cad::BaseProjection< Settings >::vars ( ) const [inline], [inherited]

Returns the variables used for projection.

0.15.372.3 Field Documentation

0.15.372.3.1 mConstraints template<typename Settings >

const Constraints& smtrat::cad::BaseProjection< Settings >::mConstraints [protected], [inherited]

0.15.372.3.2 mInfo template<typename Settings >

ProjectionInformation smtrat::cad::BaseProjection< Settings >::mInfo [protected], [inherited]

Additional info on projection, projection levels and projection polynomials.

0.15.372.3.3 mLiftingQueues template<typename Settings >

std::vector<PolynomialLiftingQueue<BaseProjection> > smtrat::cad::BaseProjection< Settings >::mLiftingQueues [protected], [inherited]

List of lifting queues that can be used for incremental projection.

0.15.372.3.4 mOperator template<typename Settings >

ProjectionOperator smtrat::cad::BaseProjection< Settings >::mOperator [protected], [inherited]

The projection operator.

0.15.372.3.5 mRemoveCallback template<typename Settings >

std::function<void(std::size_t, const SampleLiftedWith&)> smtrat::cad::BaseProjection< Settings >::mRemoveCallback [protected], [inherited]

Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.

0.15.373 smtrat::qe::cad::Projection< Settings > Class Template Reference

#include <Projection.h>

Public Member Functions

- Projection (const Constraints &c)
- void reset ()
- carl::Bitset addPolynomial (const UPoly &p, std::size_t cid, bool) override
- void removePolynomial (const UPoly &, std::size_t cid, bool) override
- bool testProjectionFactor (std::size_t level, std::size_t id) const
- void removeProjectionFactor (std::size_t level, std::size_t id)
- std::size_t size (std::size_t level) const override
- bool empty (std::size_t level) const override
- bool hasPolynomialById (std::size_t level, std::size_t id) const override
- const UPoly & getPolynomialById (std::size_t level, std::size_t id) const override

Retrieves a polynomial from its id.

- std::size_t dim () const

  Returns the dimension of the projection.
- virtual std::size_t size (std::size_t level) const=0
- std::size_t size () const
- std::size_t dim () const

  Returns the dimension of the projection.
- const auto & vars () const

  Returns the variables used for projection.
- template<typename F >
  void setRemoveCallback (F &&f)

  Sets a callback that is called whenever polynomials are removed.
- carl::Bitset addPolynomial (const Poly &p, std::size_t cid, bool isBound)

  Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.
- virtual carl::Bitset addPolynomial (const UPoly &p, std::size_t cid, bool isBound)=0

  Adds the given polynomial to the projection.
- carl::Bitset addEqConstraint (const Poly &p, std::size_t cid, bool isBound)

  Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.
- virtual carl::Bitset addEqConstraint (const UPoly &p, std::size_t cid, bool isBound)

  Adds the given polynomial of an equational constraint to the projection.
- void removePolynomial (const Poly &p, std::size_t cid, bool isBound)

  Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.
- virtual void removePolynomial (const UPoly &p, std::size_t cid, bool isBound)=0

  Removes the given polynomial from the projection.
- std::size_t size () const
- virtual bool empty ()
- OptionalID getPolyForLifting (std::size_t level, SampleLiftedWith &slw)

  Get a polynomial from this level suited for lifting.
- virtual void exportAsDot (std::ostream &) const
- virtual Origin getOrigin (std::size_t level, std::size_t id) const

Protected Member Functions

- void callRemoveCallback (std::size_t level, const SampleLiftedWith &slw) const
- std::size_t getID (std::size_t level)

  Returns a fresh polynomial id for the given level.
- void freeID (std::size_t level, std::size_t id)

  Frees a currently used polynomial id for the given level.
- carl::Variable var (std::size_t level) const

  Returns the variable that corresponds to the given level, that is the variable eliminated in this level.
- bool canBePurgedByBounds (const UPoly &p) const

  Checks whether a polynomial can safely be ignored due to the bounds.
- bool isPurged (std::size_t level, std::size_t id)

Protected Attributes

- const Constraints & mConstraints
- ProjectionInformation mInfo

  Additional info on projection, projection levels and projection polynomials.
- std::function< void(std::size_t, const SampleLiftedWith &)> mRemoveCallback

  Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.

Friends

- template<typename S >
  std::ostream & operator<< (std::ostream &os, const Projection< S > &p)

0.15.373.1 Constructor & Destructor Documentation

0.15.373.1.1 Projection() template<typename Settings >

```
smtrat::qe::cad::Projection< Settings >::Projection (
            const Constraints & c )  [inline]
```

0.15.373.2 Member Function Documentation

0.15.373.2.1 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.

0.15.373.2.2 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 >.

0.15.373.2.3 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.

0.15.373.2.4 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::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >, smtrat::cad::Projection< In
smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Increme
smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, and smtrat::cad::Projection< Incrementality::SIMPLE

0.15.373.2.5 addPolynomial() [3/3] template<typename Settings >

```
carl::Bitset smtrat::qe::cad::Projection< Settings >::addPolynomial (
            const UPoly & p,
```

```
            std::size_t cid,
            bool  )  [inline], [override]
```

0.15.373.2.6 callRemoveCallback()

```
template<typename Settings >
void smtrat::cad::BaseProjection< Settings >::callRemoveCallback (
            std::size_t level,
            const SampleLiftedWith & slw ) const  [inline], [protected], [inherited]
```

0.15.373.2.7 canBePurgedByBounds()

```
template<typename Settings >
bool smtrat::cad::BaseProjection< Settings >::canBePurgedByBounds (
            const UPoly & p ) const  [inline], [protected], [inherited]
```

Checks whether a polynomial can safely be ignored due to the bounds.

0.15.373.2.8 dim() [1/2]

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::dim ( ) const  [inline], [inherited]
```

Returns the dimension of the projection.

0.15.373.2.9 dim() [2/2]

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::dim  [inline]
```

Returns the dimension of the projection.

0.15.373.2.10 empty() [1/2]

```
template<typename Settings >
virtual bool smtrat::cad::BaseProjection< Settings >::empty ( )  [inline], [virtual], [inherited]
```

0.15.373.2.11 empty() [2/2]

```
template<typename Settings >
bool smtrat::qe::cad::Projection< Settings >::empty (
            std::size_t level ) const  [inline], [override], [virtual]
```

Implements smtrat::cad::BaseProjection< Settings >.

0.15.373.2.12 exportAsDot()

```
template<typename Settings >
virtual void smtrat::cad::BaseProjection< Settings >::exportAsDot (
            std::ostream &  ) const  [inline], [virtual], [inherited]
```

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, and smtrat::cad::Projection< Incrementality::FULL,

0.15.373.2.13 freeID()

```
template<typename Settings >
void smtrat::cad::BaseProjection< Settings >::freeID (
            std::size_t level,
            std::size_t id )  [inline], [protected], [inherited]
```

Frees a currently used polynomial id for the given level.

0.15.373.2.14 getID()

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::getID (
            std::size_t level )  [inline], [protected], [inherited]
```

Returns a fresh polynomial id for the given level.

0.15.373.2.15 getOrigin() template<typename Settings >

```
virtual Origin smtrat::cad::BaseProjection< Settings >::getOrigin (
            std::size_t level,
            std::size_t id ) const [inline], [virtual], [inherited]
```

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, BT, Settings >.

0.15.373.2.16 getPolyForLifting() template<typename Settings >

```
OptionalID smtrat::cad::BaseProjection< Settings >::getPolyForLifting (
            std::size_t level,
            SampleLiftedWith & slw ) [inline], [inherited]
```

Get a polynomial from this level suited for lifting.

0.15.373.2.17 getPolynomialById() template<typename Settings >

```
const UPoly& smtrat::qe::cad::Projection< Settings >::getPolynomialById (
            std::size_t level,
            std::size_t id ) const [inline], [override], [virtual]
```

Retrieves a polynomial from its id.
Implements smtrat::cad::BaseProjection< Settings >.

0.15.373.2.18 hasPolynomialById() template<typename Settings >

```
bool smtrat::qe::cad::Projection< Settings >::hasPolynomialById (
            std::size_t level,
            std::size_t id ) const [inline], [override], [virtual]
```

Implements smtrat::cad::BaseProjection< Settings >.

0.15.373.2.19 isPurged() template<typename Settings >

```
bool smtrat::cad::BaseProjection< Settings >::isPurged (
            std::size_t level,
            std::size_t id ) [inline], [protected], [inherited]
```

0.15.373.2.20 removePolynomial() [1/3] template<typename Settings >

```
void smtrat::cad::BaseProjection< Settings >::removePolynomial (
            const Poly & p,
            std::size_t cid,
            bool isBound ) [inline], [inherited]
```

Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.

0.15.373.2.21 removePolynomial() [2/3] template<typename Settings >

```
void smtrat::qe::cad::Projection< Settings >::removePolynomial (
            const UPoly & ,
            std::size_t cid,
            bool ) [inline], [override]
```

0.15.373.2.22 removePolynomial() [3/3] template<typename Settings >

```
virtual void smtrat::cad::BaseProjection< Settings >::removePolynomial (
            const UPoly & p,
            std::size_t cid,
            bool isBound ) [pure virtual], [inherited]
```

Removes the given polynomial from the projection.

Implemented in smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::ModelBasedProje
smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, S
smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >, and smtrat::cad::Projection< Incrementality::NONE, Backtracking::

0.15.373.2.23 removeProjectionFactor()

```
template<typename Settings >
void smtrat::qe::cad::Projection< Settings >::removeProjectionFactor (
            std::size_t level,
            std::size_t id )  [inline]
```

0.15.373.2.24 reset()

```
template<typename Settings >
void smtrat::qe::cad::Projection< Settings >::reset ( )  [inline]
```

0.15.373.2.25 setRemoveCallback()

```
template<typename Settings >
template<typename F >
void smtrat::cad::BaseProjection< Settings >::setRemoveCallback (
            F && f )  [inline], [inherited]
```

Sets a callback that is called whenever polynomials are removed.

0.15.373.2.26 size() [1/4]

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::size ( ) const  [inline], [inherited]
```

0.15.373.2.27 size() [2/4]

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::size (
            void  )  [inline]
```

0.15.373.2.28 size() [3/4]

```
template<typename Settings >
std::size_t smtrat::qe::cad::Projection< Settings >::size (
            std::size_t level ) const  [inline], [override], [virtual]
```

Implements smtrat::cad::BaseProjection< Settings >.

0.15.373.2.29 size() [4/4]

```
template<typename Settings >
virtual std::size_t smtrat::cad::BaseProjection< Settings >::size (
            void  )
```

0.15.373.2.30 testProjectionFactor()

```
template<typename Settings >
bool smtrat::qe::cad::Projection< Settings >::testProjectionFactor (
            std::size_t level,
            std::size_t id ) const  [inline]
```

0.15.373.2.31 var()

```
template<typename Settings >
carl::Variable smtrat::cad::BaseProjection< Settings >::var (
            std::size_t level ) const  [inline], [protected], [inherited]
```

Returns the variable that corresponds to the given level, that is the variable eliminated in this level.

0.15.373.2.32 vars() template<typename Settings >

const auto& smtrat::cad::BaseProjection< Settings >::vars ( ) const [inline], [inherited]

Returns the variables used for projection.

0.15.373.3 Friends And Related Function Documentation

0.15.373.3.1 operator<< template<typename Settings >

```
template<typename S >
std::ostream& operator<< (
            std::ostream & os,
            const Projection< S > & p ) [friend]
```

0.15.373.4 Field Documentation

0.15.373.4.1 mConstraints template<typename Settings >

const Constraints& smtrat::cad::BaseProjection< Settings >::mConstraints [protected], [inherited]

0.15.373.4.2 mInfo template<typename Settings >

ProjectionInformation smtrat::cad::BaseProjection< Settings >::mInfo [protected], [inherited]

Additional info on projection, projection levels and projection polynomials.

0.15.373.4.3 mRemoveCallback template<typename Settings >

std::function<void(std::size_t, const SampleLiftedWith&)> smtrat::cad::BaseProjection< Settings >::mRemoveCallback [protected], [inherited]

Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.

0.15.374 smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings > Class Template Reference

#include <Projection_EC.h>

Public Member Functions

- Projection (const Constraints &c)
- void reset ()
- carl::Bitset addPolynomial (const UPoly &p, std::size_t cid, bool isBound) override

  Adds the given polynomial to the projection.
- carl::Bitset addEqConstraint (const UPoly &p, std::size_t cid, bool isBound) override

  Adds the given polynomial of an equational constraint to the projection.
- void removePolynomial (const UPoly &p, std::size_t cid, bool isBound) override

  Removes the given polynomial from the projection.
- std::size_t size (std::size_t level) const override
- bool empty (std::size_t level) const override
- carl::Bitset projectNewPolynomial (const ConstraintSelection &cs=carl::Bitset(true))
- bool hasPolynomialById (std::size_t level, std::size_t id) const override
- const UPoly & getPolynomialById (std::size_t level, std::size_t id) const override

  Retrieves a polynomial from its id.
- void exportAsDot (std::ostream &out) const override
- void printPolynomialIDs () const

- std::size_t dim () const

  Returns the dimension of the projection.
- virtual std::size_t size (std::size_t level) const=0
- std::size_t size () const
- std::size_t dim () const

  Returns the dimension of the projection.
- const auto & vars () const

  Returns the variables used for projection.
- template<typename F >
  void setRemoveCallback (F &&f)

  Sets a callback that is called whenever polynomials are removed.
- carl::Bitset addPolynomial (const Poly &p, std::size_t cid, bool isBound)

  Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.
- carl::Bitset addEqConstraint (const Poly &p, std::size_t cid, bool isBound)

  Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.
- void removePolynomial (const Poly &p, std::size_t cid, bool isBound)

  Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.
- std::size_t size () const
- virtual bool empty ()
- OptionalID getPolyForLifting (std::size_t level, SampleLiftedWith &slw)

  Get a polynomial from this level suited for lifting.
- virtual Origin getOrigin (std::size_t level, std::size_t id) const

Protected Member Functions

- void callRemoveCallback (std::size_t level, const SampleLiftedWith &slw) const
- std::size_t getID (std::size_t level)

  Returns a fresh polynomial id for the given level.
- void freeID (std::size_t level, std::size_t id)

  Frees a currently used polynomial id for the given level.
- carl::Variable var (std::size_t level) const

  Returns the variable that corresponds to the given level, that is the variable eliminated in this level.
- bool canBePurgedByBounds (const UPoly &p) const

  Checks whether a polynomial can safely be ignored due to the bounds.
- bool isPurged (std::size_t level, std::size_t id)

Protected Attributes

- std::function< void(std::size_t, const SampleLiftedWith &)> mRemoveCallback

  Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.

Friends

- template<typename S >
  std::ostream & operator<< (std::ostream &os, const Projection< Incrementality::FULL, Backtracking::HIDE, S > &p)

0.15.374.1 Constructor & Destructor Documentation

0.15.374.1.1 Projection() template<typename Settings >

```
smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >::Projection (
            const Constraints & c )  [inline]
```

0.15.374.2 Member Function Documentation

0.15.374.2.1 addEqConstraint() [1/2] template<typename Settings >

```
carl::Bitset smtrat::cad::BaseProjection< Settings >::addEqConstraint (
            const Poly & p,
            std::size_t cid,
            bool isBound )  [inline], [inherited]
```

Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.

0.15.374.2.2 addEqConstraint() [2/2] template<typename Settings >

```
carl::Bitset smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >↩
::addEqConstraint (
            const UPoly & p,
            std::size_t cid,
            bool isBound )  [inline], [override], [virtual]
```

Adds the given polynomial of an equational constraint to the projection.
Polynomial already exists.
Reimplemented from smtrat::cad::BaseProjection< Settings >.

0.15.374.2.3 addPolynomial() [1/2] template<typename Settings >

```
carl::Bitset smtrat::cad::BaseProjection< Settings >::addPolynomial (
            const Poly & p,
            std::size_t cid,
            bool isBound )  [inline], [inherited]
```

Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.

0.15.374.2.4 addPolynomial() [2/2] template<typename Settings >

```
carl::Bitset smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >↩
::addPolynomial (
            const UPoly & p,
            std::size_t cid,
            bool isBound )  [inline], [override], [virtual]
```

Adds the given polynomial to the projection.
Implements smtrat::cad::BaseProjection< Settings >.

0.15.374.2.5 callRemoveCallback() template<typename Settings >

```
void smtrat::cad::BaseProjection< Settings >::callRemoveCallback (
            std::size_t level,
            const SampleLiftedWith & slw ) const  [inline], [protected], [inherited]
```

0.15.374.2.6 canBePurgedByBounds() template<typename Settings >

```
bool smtrat::cad::BaseProjection< Settings >::canBePurgedByBounds (
            const UPoly & p ) const  [inline], [protected], [inherited]
```

Checks whether a polynomial can safely be ignored due to the bounds.

0.15.374.2.7 dim() [1/2] template<typename Settings >

```
std::size_t smtrat::cad::BaseProjection< Settings >::dim ( ) const [inline], [inherited]
```

Returns the dimension of the projection.

0.15.374.2.8 dim() [2/2] template<typename Settings >

```
std::size_t smtrat::cad::BaseProjection< Settings >::dim [inline]
```

Returns the dimension of the projection.

0.15.374.2.9 empty() [1/2] template<typename Settings >

```
virtual bool smtrat::cad::BaseProjection< Settings >::empty ( ) [inline], [virtual], [inherited]
```

0.15.374.2.10 empty() [2/2] template<typename Settings >

```
bool smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >::empty (
            std::size_t level ) const [inline], [override], [virtual]
```

Implements smtrat::cad::BaseProjection< Settings >.

0.15.374.2.11 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 >.

0.15.374.2.12 freeID() template<typename Settings >

```
void smtrat::cad::BaseProjection< Settings >::freeID (
            std::size_t level,
            std::size_t id ) [inline], [protected], [inherited]
```

Frees a currently used polynomial id for the given level.

0.15.374.2.13 getID() template<typename Settings >

```
std::size_t smtrat::cad::BaseProjection< Settings >::getID (
            std::size_t level ) [inline], [protected], [inherited]
```

Returns a fresh polynomial id for the given level.

0.15.374.2.14 getOrigin() template<typename Settings >

```
virtual Origin smtrat::cad::BaseProjection< Settings >::getOrigin (
            std::size_t level,
            std::size_t id ) const [inline], [virtual], [inherited]
```

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, BT, Settings >.

0.15.374.2.15 getPolyForLifting() template<typename Settings >

```
OptionalID smtrat::cad::BaseProjection< Settings >::getPolyForLifting (
            std::size_t level,
            SampleLiftedWith & slw ) [inline], [inherited]
```

Get a polynomial from this level suited for lifting.

0.15.374.2.16 getPolynomialById() template<typename Settings >

```
const UPoly& smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >↩
::getPolynomialById (
            std::size_t level,
            std::size_t id ) const  [inline], [override], [virtual]
```

Retrieves a polynomial from its id.
Implements smtrat::cad::BaseProjection< Settings >.

0.15.374.2.17 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 >.

0.15.374.2.18 isPurged() template<typename Settings >

```
bool smtrat::cad::BaseProjection< Settings >::isPurged (
            std::size_t level,
            std::size_t id )  [inline], [protected], [inherited]
```

0.15.374.2.19 printPolynomialIDs() template<typename Settings >

```
void smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >::print↩
PolynomialIDs ( ) const  [inline]
```

0.15.374.2.20 projectNewPolynomial() template<typename Settings >

```
carl::Bitset smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >↩
::projectNewPolynomial (
            const ConstraintSelection & cs = carl::Bitset(true) )  [inline]
```

0.15.374.2.21 removePolynomial() [1/2] template<typename Settings >

```
void smtrat::cad::BaseProjection< Settings >::removePolynomial (
            const Poly & p,
            std::size_t cid,
            bool isBound )  [inline], [inherited]
```

Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.

0.15.374.2.22 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 >.

0.15.374.2.23 reset() template<typename Settings >

```
void smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >::reset ( )
[inline]
```

0.15.374.2.24 setRemoveCallback()

```
template<typename Settings >
template<typename F >
void smtrat::cad::BaseProjection< Settings >::setRemoveCallback (
            F && f )  [inline], [inherited]
```

Sets a callback that is called whenever polynomials are removed.

0.15.374.2.25 size() [1/4]

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::size ( ) const  [inline], [inherited]
```

0.15.374.2.26 size() [2/4]

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::size (
            void  )  [inline]
```

0.15.374.2.27 size() [3/4]

```
template<typename Settings >
std::size_t smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >↩
::size (
            std::size_t level ) const  [inline], [override], [virtual]
```

Implements smtrat::cad::BaseProjection< Settings >.

0.15.374.2.28 size() [4/4]

```
template<typename Settings >
virtual std::size_t smtrat::cad::BaseProjection< Settings >::size (
            void  )
```

0.15.374.2.29 var()

```
template<typename Settings >
carl::Variable smtrat::cad::BaseProjection< Settings >::var (
            std::size_t level ) const  [inline], [protected], [inherited]
```

Returns the variable that corresponds to the given level, that is the variable eliminated in this level.

0.15.374.2.30 vars()

```
template<typename Settings >
const auto& smtrat::cad::BaseProjection< Settings >::vars ( ) const  [inline], [inherited]
```

Returns the variables used for projection.

0.15.374.3 Friends And Related Function Documentation

0.15.374.3.1 operator<<

```
template<typename Settings >
template<typename S >
std::ostream& operator<< (
            std::ostream & os,
            const Projection< Incrementality::FULL, Backtracking::HIDE, S > & p )  [friend]
```

0.15.374.4 Field Documentation

0.15.374.4.1 mRemoveCallback

```
template<typename Settings >
std::function<void(std::size_t, const SampleLiftedWith&)> smtrat::cad::BaseProjection< Settings
>::mRemoveCallback  [protected], [inherited]
```

Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.

0.15.375 smtrat::cad::Projection< Incrementality::FULL, BT, Settings > Class Template Reference

```
#include <Projection_F.h>
```

Public Member Functions

- Projection (const Constraints &c)
- void reset ()
- carl::Bitset addPolynomial (const UPoly &p, std::size_t cid, bool isBound) override

  Adds the given polynomial to the projection.
- void removePolynomial (const UPoly &p, std::size_t cid, bool isBound) override

  Removes the given polynomial from the projection.
- std::size_t size (std::size_t level) const override
- bool empty (std::size_t level) const override
- carl::Bitset projectNewPolynomial (const ConstraintSelection &cs=carl::Bitset(true))
- bool hasPolynomialById (std::size_t level, std::size_t id) const override
- const UPoly & getPolynomialById (std::size_t level, std::size_t id) const override

  Retrieves a polynomial from its id.
- void exportAsDot (std::ostream &out) const override
- Origin getOrigin (std::size_t level, std::size_t id) const override
- std::size_t dim () const

  Returns the dimension of the projection.
- virtual std::size_t size (std::size_t level) const=0
- std::size_t size () const
- std::size_t dim () const

  Returns the dimension of the projection.
- const auto & vars () const

  Returns the variables used for projection.
- template<typename F >
  void setRemoveCallback (F &&f)

  Sets a callback that is called whenever polynomials are removed.
- carl::Bitset addPolynomial (const Poly &p, std::size_t cid, bool isBound)

  Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.
- carl::Bitset addEqConstraint (const Poly &p, std::size_t cid, bool isBound)

  Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.
- virtual carl::Bitset addEqConstraint (const UPoly &p, std::size_t cid, bool isBound)

  Adds the given polynomial of an equational constraint to the projection.
- void removePolynomial (const Poly &p, std::size_t cid, bool isBound)

  Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.
- std::size_t size () const
- virtual bool empty ()
- OptionalID getPolyForLifting (std::size_t level, SampleLiftedWith &slw)

  Get a polynomial from this level suited for lifting.

Protected Member Functions

- void callRemoveCallback (std::size_t level, const SampleLiftedWith &slw) const
- std::size_t getID (std::size_t level)

  Returns a fresh polynomial id for the given level.
- void freeID (std::size_t level, std::size_t id)

  Frees a currently used polynomial id for the given level.
- carl::Variable var (std::size_t level) const

Returns the variable that corresponds to the given level, that is the variable eliminated in this level.

- bool canBePurgedByBounds (const UPoly &p) const

  Checks whether a polynomial can safely be ignored due to the bounds.
- bool isPurged (std::size_t level, std::size_t id)

Protected Attributes

- ProjectionInformation mInfo

  Additional info on projection, projection levels and projection polynomials.
- std::function< void(std::size_t, const SampleLiftedWith &)> mRemoveCallback

  Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.

Friends

- template<typename S , Backtracking B>
  std::ostream & operator<< (std::ostream &os, const Projection< Incrementality::FULL, B, S > &p)

0.15.375.1 Constructor & Destructor Documentation

0.15.375.1.1 Projection()

```
template<typename Settings , Backtracking BT>
smtrat::cad::Projection< Incrementality::FULL, BT, Settings >::Projection (
            const Constraints & c )  [inline]
```

0.15.375.2 Member Function Documentation

0.15.375.2.1 addEqConstraint() [1/2]

```
template<typename Settings >
carl::Bitset smtrat::cad::BaseProjection< Settings >::addEqConstraint (
            const Poly & p,
            std::size_t cid,
            bool isBound )  [inline], [inherited]
```

Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.

0.15.375.2.2 addEqConstraint() [2/2]

```
template<typename Settings >
virtual carl::Bitset smtrat::cad::BaseProjection< Settings >::addEqConstraint (
            const UPoly & p,
            std::size_t cid,
            bool isBound )  [inline], [virtual], [inherited]
```

Adds the given polynomial of an equational constraint to the projection.
Reimplemented in smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >.

0.15.375.2.3 addPolynomial() [1/2]

```
template<typename Settings >
carl::Bitset smtrat::cad::BaseProjection< Settings >::addPolynomial (
            const Poly & p,
            std::size_t cid,
            bool isBound )  [inline], [inherited]
```

Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.

0.15.375.2.4 addPolynomial() [2/2] template<typename Settings , Backtracking BT>

```
carl::Bitset smtrat::cad::Projection< Incrementality::FULL, BT, Settings >::addPolynomial (
            const UPoly & p,
            std::size_t cid,
            bool isBound )  [inline], [override], [virtual]
```

Adds the given polynomial to the projection.
Implements smtrat::cad::BaseProjection< Settings >.

0.15.375.2.5 callRemoveCallback() template<typename Settings >

```
void smtrat::cad::BaseProjection< Settings >::callRemoveCallback (
            std::size_t level,
            const SampleLiftedWith & slw ) const  [inline], [protected], [inherited]
```

0.15.375.2.6 canBePurgedByBounds() template<typename Settings >

```
bool smtrat::cad::BaseProjection< Settings >::canBePurgedByBounds (
            const UPoly & p ) const  [inline], [protected], [inherited]
```

Checks whether a polynomial can safely be ignored due to the bounds.

0.15.375.2.7 dim() [1/2] template<typename Settings >

```
std::size_t smtrat::cad::BaseProjection< Settings >::dim ( ) const  [inline], [inherited]
```

Returns the dimension of the projection.

0.15.375.2.8 dim() [2/2] template<typename Settings , Backtracking BT>

```
std::size_t smtrat::cad::BaseProjection< Settings >::dim  [inline]
```

Returns the dimension of the projection.

0.15.375.2.9 empty() [1/2] template<typename Settings >

```
virtual bool smtrat::cad::BaseProjection< Settings >::empty ( )  [inline], [virtual], [inherited]
```

0.15.375.2.10 empty() [2/2] template<typename Settings , Backtracking BT>

```
bool smtrat::cad::Projection< Incrementality::FULL, BT, Settings >::empty (
            std::size_t level ) const  [inline], [override], [virtual]
```

Implements smtrat::cad::BaseProjection< Settings >.

0.15.375.2.11 exportAsDot() template<typename Settings , Backtracking BT>

```
void smtrat::cad::Projection< Incrementality::FULL, BT, Settings >::exportAsDot (
            std::ostream & out ) const  [inline], [override], [virtual]
```

Reimplemented from smtrat::cad::BaseProjection< Settings >.

0.15.375.2.12 freeID() template<typename Settings >

```
void smtrat::cad::BaseProjection< Settings >::freeID (
            std::size_t level,
            std::size_t id )  [inline], [protected], [inherited]
```

Frees a currently used polynomial id for the given level.

0.15.375.2.13 getID()

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::getID (
            std::size_t level ) [inline], [protected], [inherited]
```

Returns a fresh polynomial id for the given level.

0.15.375.2.14 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 >.

0.15.375.2.15 getPolyForLifting()

```
template<typename Settings >
OptionalID smtrat::cad::BaseProjection< Settings >::getPolyForLifting (
            std::size_t level,
            SampleLiftedWith & slw ) [inline], [inherited]
```

Get a polynomial from this level suited for lifting.

0.15.375.2.16 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 >.

0.15.375.2.17 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 >.

0.15.375.2.18 isPurged()

```
template<typename Settings >
bool smtrat::cad::BaseProjection< Settings >::isPurged (
            std::size_t level,
            std::size_t id ) [inline], [protected], [inherited]
```

0.15.375.2.19 projectNewPolynomial()

```
template<typename Settings , Backtracking BT>
carl::Bitset smtrat::cad::Projection< Incrementality::FULL, BT, Settings >::projectNewPolynomial
(
            const ConstraintSelection & cs = carl::Bitset(true) ) [inline]
```

0.15.375.2.20 removePolynomial() [1/2]

```
template<typename Settings >
void smtrat::cad::BaseProjection< Settings >::removePolynomial (
            const Poly & p,
            std::size_t cid,
            bool isBound ) [inline], [inherited]
```

Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.

0.15.375.2.21 removePolynomial() [2/2]

```
template<typename Settings , Backtracking BT>
void smtrat::cad::Projection< Incrementality::FULL, BT, Settings >::removePolynomial (
            const UPoly & p,
            std::size_t cid,
            bool isBound )  [inline], [override], [virtual]
```

Removes the given polynomial from the projection.
Implements smtrat::cad::BaseProjection< Settings >.

0.15.375.2.22 reset()

```
template<typename Settings , Backtracking BT>
void smtrat::cad::Projection< Incrementality::FULL, BT, Settings >::reset ( )  [inline]
```

0.15.375.2.23 setRemoveCallback()

```
template<typename Settings >
template<typename F >
void smtrat::cad::BaseProjection< Settings >::setRemoveCallback (
            F && f )  [inline], [inherited]
```

Sets a callback that is called whenever polynomials are removed.

0.15.375.2.24 size() [1/4]

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::size ( ) const  [inline], [inherited]
```

0.15.375.2.25 size() [2/4]

```
template<typename Settings , Backtracking BT>
std::size_t smtrat::cad::BaseProjection< Settings >::size (
            void )  [inline]
```

0.15.375.2.26 size() [3/4]

```
template<typename Settings , Backtracking BT>
std::size_t smtrat::cad::Projection< Incrementality::FULL, BT, Settings >::size (
            std::size_t level ) const  [inline], [override], [virtual]
```

Implements smtrat::cad::BaseProjection< Settings >.

0.15.375.2.27 size() [4/4]

```
template<typename Settings , Backtracking BT>
virtual std::size_t smtrat::cad::BaseProjection< Settings >::size (
            void )
```

0.15.375.2.28 var()

```
template<typename Settings >
carl::Variable smtrat::cad::BaseProjection< Settings >::var (
            std::size_t level ) const  [inline], [protected], [inherited]
```

Returns the variable that corresponds to the given level, that is the variable eliminated in this level.

0.15.375.2.29 vars()

```
template<typename Settings >
const auto& smtrat::cad::BaseProjection< Settings >::vars ( ) const  [inline], [inherited]
```

Returns the variables used for projection.

0.15.375.3 Friends And Related Function Documentation

0.15.375.3.1 operator<< template<typename Settings , Backtracking BT>

```
template<typename S , Backtracking B>
std::ostream& operator<< (
            std::ostream & os,
            const Projection< Incrementality::FULL, B, S > & p )  [friend]
```

0.15.375.4 Field Documentation

0.15.375.4.1 mInfo template<typename Settings >

```
ProjectionInformation smtrat::cad::BaseProjection< Settings >::mInfo  [protected], [inherited]
```

Additional info on projection, projection levels and projection polynomials.

0.15.375.4.2 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.

0.15.376 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)

0.15.376.1 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.

0.15.376.2 Constructor & Destructor Documentation

0.15.376.2.1 Projection() template<typename Settings >

```
smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >::Projection (
            const Constraints & c )  [inline]
```

0.15.376.3 Member Function Documentation

0.15.376.3.1 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.

0.15.376.3.2 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 >.

0.15.376.3.3 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.

0.15.376.3.4 addPolynomial() [2/2] template<typename Settings >

```
carl::Bitset smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings
>::addPolynomial (
            const UPoly & p,
            std::size_t cid,
            bool  )  [inline], [override], [virtual]
```

Adds the given polynomial to the projection with the given constraint id as origin.
Asserts that the main variable of the polynomial is the first variable.
Implements smtrat::cad::BaseProjection< Settings >.

0.15.376.3.5 callRemoveCallback()

```
template<typename Settings >
void smtrat::cad::BaseProjection< Settings >::callRemoveCallback (
            std::size_t level,
            const SampleLiftedWith & slw ) const [inline], [protected], [inherited]
```

0.15.376.3.6 canBePurgedByBounds()

```
template<typename Settings >
bool smtrat::cad::BaseProjection< Settings >::canBePurgedByBounds (
            const UPoly & p ) const [inline], [protected], [inherited]
```

Checks whether a polynomial can safely be ignored due to the bounds.

0.15.376.3.7 dim() [1/2]

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::dim ( ) const [inline], [inherited]
```

Returns the dimension of the projection.

0.15.376.3.8 dim() [2/2]

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::dim [inline]
```

Returns the dimension of the projection.

0.15.376.3.9 empty() [1/2]

```
template<typename Settings >
virtual bool smtrat::cad::BaseProjection< Settings >::empty ( ) [inline], [virtual], [inherited]
```

0.15.376.3.10 empty() [2/2]

```
template<typename Settings >
bool smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >::empty (
            std::size_t level ) const [inline], [override], [virtual]
```

Returns whether the number of polynomials in this level is zero.

Implements smtrat::cad::BaseProjection< Settings >.

0.15.376.3.11 exportAsDot()

```
template<typename Settings >
virtual void smtrat::cad::BaseProjection< Settings >::exportAsDot (
            std::ostream & ) const [inline], [virtual], [inherited]
```

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, and smtrat::cad::Projection< Incrementality::FULL,

0.15.376.3.12 freeID()

```
template<typename Settings >
void smtrat::cad::BaseProjection< Settings >::freeID (
            std::size_t level,
            std::size_t id ) [inline], [protected], [inherited]
```

Frees a currently used polynomial id for the given level.

0.15.376.3.13 getID()

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::getID (
            std::size_t level ) [inline], [protected], [inherited]
```

Returns a fresh polynomial id for the given level.

0.15.376.3.14 getOrigin()

```
template<typename Settings >
virtual Origin smtrat::cad::BaseProjection< Settings >::getOrigin (
            std::size_t level,
            std::size_t id ) const [inline], [virtual], [inherited]
```

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, BT, Settings >.

0.15.376.3.15 getPolyForLifting() template<typename Settings >

```
OptionalID smtrat::cad::BaseProjection< Settings >::getPolyForLifting (
            std::size_t level,
            SampleLiftedWith & slw )  [inline], [inherited]
```

Get a polynomial from this level suited for lifting.

0.15.376.3.16 getPolynomialById() template<typename Settings >

```
const UPoly& smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings
>::getPolynomialById (
            std::size_t level,
            std::size_t id ) const  [inline], [override], [virtual]
```

Get the polynomial from this level with the given id.
Implements smtrat::cad::BaseProjection< Settings >.

0.15.376.3.17 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 >.

0.15.376.3.18 isPurged() template<typename Settings >

```
bool smtrat::cad::BaseProjection< Settings >::isPurged (
            std::size_t level,
            std::size_t id )  [inline], [protected], [inherited]
```

0.15.376.3.19 projectNewPolynomial() template<typename Settings >

```
carl::Bitset smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings
>::projectNewPolynomial (
            const ConstraintSelection &  = carl::Bitset(true) )  [inline]
```

Returns false, as the projection is not incremental.

0.15.376.3.20 removePolynomial() [1/2] template<typename Settings >

```
void smtrat::cad::BaseProjection< Settings >::removePolynomial (
            const Poly & p,
            std::size_t cid,
            bool isBound )  [inline], [inherited]
```

Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.

0.15.376.3.21 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 >.

0.15.376.3.22 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.

0.15.376.3.23 setRemoveCallback()

```
template<typename Settings >
template<typename F >
void smtrat::cad::BaseProjection< Settings >::setRemoveCallback (
            F && f ) [inline], [inherited]
```

Sets a callback that is called whenever polynomials are removed.

0.15.376.3.24 size() [1/4]

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::size ( ) const [inline], [inherited]
```

0.15.376.3.25 size() [2/4]

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::size (
            void ) [inline]
```

0.15.376.3.26 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 >.

0.15.376.3.27 size() [4/4]

```
template<typename Settings >
virtual std::size_t smtrat::cad::BaseProjection< Settings >::size (
            void )
```

0.15.376.3.28 var()

```
template<typename Settings >
carl::Variable smtrat::cad::BaseProjection< Settings >::var (
            std::size_t level ) const [inline], [protected], [inherited]
```

Returns the variable that corresponds to the given level, that is the variable eliminated in this level.

0.15.376.3.29 vars()

```
template<typename Settings >
const auto& smtrat::cad::BaseProjection< Settings >::vars ( ) const [inline], [inherited]
```

Returns the variables used for projection.

0.15.376.4 Friends And Related Function Documentation

0.15.376.4.1 operator<<

```
template<typename Settings >
template<typename S >
std::ostream& operator<< (
            std::ostream & os,
            const Projection< Incrementality::NONE, Backtracking::ORDERED, S > & p ) [friend]
```

0.15.376.5 Field Documentation

0.15.376.5.1 mConstraints `template<typename Settings >`

`const Constraints& smtrat::cad::BaseProjection< Settings >::mConstraints [protected], [inherited]`

0.15.376.5.2 mInfo `template<typename Settings >`

`ProjectionInformation smtrat::cad::BaseProjection< Settings >::mInfo [protected], [inherited]`

Additional info on projection, projection levels and projection polynomials.

0.15.376.5.3 mRemoveCallback `template<typename Settings >`

`std::function<void(std::size_t, const SampleLiftedWith&)> smtrat::cad::BaseProjection< Settings >::mRemoveCallback [protected], [inherited]`

Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.

0.15.377 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)

0.15.377.1 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.

0.15.377.2 Constructor & Destructor Documentation

0.15.377.2.1 Projection()

```
template<typename Settings >
smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >::Projection
(
            const Constraints & c )  [inline]
```

0.15.377.3 Member Function Documentation

0.15.377.3.1 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.

0.15.377.3.2 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 >.

0.15.377.3.3 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.

0.15.377.3.4 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 >.

0.15.377.3.5 callRemoveCallback() template<typename Settings >

```
void smtrat::cad::BaseProjection< Settings >::callRemoveCallback (
            std::size_t level,
            const SampleLiftedWith & slw ) const  [inline], [protected], [inherited]
```

0.15.377.3.6 canBePurgedByBounds() template<typename Settings >

```
bool smtrat::cad::BaseProjection< Settings >::canBePurgedByBounds (
            const UPoly & p ) const  [inline], [protected], [inherited]
```

Checks whether a polynomial can safely be ignored due to the bounds.

0.15.377.3.7 dim() [1/2] template<typename Settings >

```
std::size_t smtrat::cad::BaseProjection< Settings >::dim ( ) const  [inline], [inherited]
```

Returns the dimension of the projection.

0.15.377.3.8 dim() [2/2] template<typename Settings >

```
std::size_t smtrat::cad::BaseProjection< Settings >::dim [inline]
```

Returns the dimension of the projection.

0.15.377.3.9 empty() [1/2] template<typename Settings >

```
virtual bool smtrat::cad::BaseProjection< Settings >::empty ( ) [inline], [virtual], [inherited]
```

0.15.377.3.10 empty() [2/2] template<typename Settings >

```
bool smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >::empty
(
            std::size_t level ) const [inline], [override], [virtual]
```

Implements smtrat::cad::BaseProjection< Settings >.

0.15.377.3.11 exportAsDot() template<typename Settings >

```
virtual void smtrat::cad::BaseProjection< Settings >::exportAsDot (
            std::ostream & ) const [inline], [virtual], [inherited]
```

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, and smtrat::cad::Projection< Incrementality::FULL,

0.15.377.3.12 freeID() template<typename Settings >

```
void smtrat::cad::BaseProjection< Settings >::freeID (
            std::size_t level,
            std::size_t id ) [inline], [protected], [inherited]
```

Frees a currently used polynomial id for the given level.

0.15.377.3.13 getID() template<typename Settings >

```
std::size_t smtrat::cad::BaseProjection< Settings >::getID (
            std::size_t level ) [inline], [protected], [inherited]
```

Returns a fresh polynomial id for the given level.

0.15.377.3.14 getOrigin() template<typename Settings >

```
virtual Origin smtrat::cad::BaseProjection< Settings >::getOrigin (
            std::size_t level,
            std::size_t id ) const [inline], [virtual], [inherited]
```

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, BT, Settings >.

0.15.377.3.15 getPolyForLifting() template<typename Settings >

```
OptionalID smtrat::cad::BaseProjection< Settings >::getPolyForLifting (
            std::size_t level,
            SampleLiftedWith & slw ) [inline], [inherited]
```

Get a polynomial from this level suited for lifting.

0.15.377.3.16 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 >.

0.15.377.3.17 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 >.

0.15.377.3.18 isPurged()

```
template<typename Settings >
bool smtrat::cad::BaseProjection< Settings >::isPurged (
            std::size_t level,
            std::size_t id )  [inline], [protected], [inherited]
```

0.15.377.3.19 projectNewPolynomial()

```
template<typename Settings >
carl::Bitset smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings
>::projectNewPolynomial (
            const ConstraintSelection &  = carl::Bitset(true) )  [inline]
```

0.15.377.3.20 removePolynomial() [1/2]

```
template<typename Settings >
void smtrat::cad::BaseProjection< Settings >::removePolynomial (
            const Poly & p,
            std::size_t cid,
            bool isBound )  [inline], [inherited]
```

Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.

0.15.377.3.21 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 >.

0.15.377.3.22 reset()

```
template<typename Settings >
void smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >::reset
( )  [inline]
```

0.15.377.3.23 setRemoveCallback()

```
template<typename Settings >
template<typename F >
void smtrat::cad::BaseProjection< Settings >::setRemoveCallback (
            F && f )  [inline], [inherited]
```

Sets a callback that is called whenever polynomials are removed.

0.15.377.3.24 size() [1/4]

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::size ( ) const  [inline], [inherited]
```

0.15.377.3.25 size() [2/4] template<typename Settings >

```
std::size_t smtrat::cad::BaseProjection< Settings >::size (
            void ) [inline]
```

0.15.377.3.26 size() [3/4] template<typename Settings >

```
std::size_t smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings
>::size (
            std::size_t level ) const [inline], [override], [virtual]
```

Implements smtrat::cad::BaseProjection< Settings >.

0.15.377.3.27 size() [4/4] template<typename Settings >

```
virtual std::size_t smtrat::cad::BaseProjection< Settings >::size (
            void )
```

0.15.377.3.28 var() template<typename Settings >

```
carl::Variable smtrat::cad::BaseProjection< Settings >::var (
            std::size_t level ) const [inline], [protected], [inherited]
```

Returns the variable that corresponds to the given level, that is the variable eliminated in this level.

0.15.377.3.29 vars() template<typename Settings >

```
const auto& smtrat::cad::BaseProjection< Settings >::vars ( ) const [inline], [inherited]
```

Returns the variables used for projection.

0.15.377.4 Friends And Related Function Documentation

0.15.377.4.1 operator<< template<typename Settings >

```
template<typename S >
std::ostream& operator<< (
            std::ostream & os,
            const Projection< Incrementality::NONE, Backtracking::UNORDERED, S > & p ) [friend]
```

0.15.377.5 Field Documentation

0.15.377.5.1 mConstraints template<typename Settings >

```
const Constraints& smtrat::cad::BaseProjection< Settings >::mConstraints [protected], [inherited]
```

0.15.377.5.2 mInfo template<typename Settings >

```
ProjectionInformation smtrat::cad::BaseProjection< Settings >::mInfo [protected], [inherited]
```

Additional info on projection, projection levels and projection polynomials.

0.15.377.5.3 mRemoveCallback template<typename Settings >

```
std::function<void(std::size_t, const SampleLiftedWith&)> smtrat::cad::BaseProjection< Settings
>::mRemoveCallback [protected], [inherited]
```

Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.

0.15.378 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)

0.15.378.1 Constructor & Destructor Documentation

0.15.378.1.1 Projection()

```
template<typename Settings , Backtracking BT>
template<typename ... Args>
smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >::Projection (
            Args &&... args ) [inline]
```

0.15.378.2 Member Function Documentation

0.15.378.2.1 addEqConstraint() [1/2]

```
template<typename Settings >
carl::Bitset smtrat::cad::BaseProjection< Settings >::addEqConstraint (
            const Poly & p,
            std::size_t cid,
            bool isBound ) [inline], [inherited]
```

Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.

0.15.378.2.2 addEqConstraint() [2/2]

```
template<typename Settings >
virtual carl::Bitset smtrat::cad::BaseProjection< Settings >::addEqConstraint (
            const UPoly & p,
            std::size_t cid,
            bool isBound ) [inline], [virtual], [inherited]
```

Adds the given polynomial of an equational constraint to the projection.
Reimplemented in smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >.

0.15.378.2.3 addPolynomial() [1/2] template<typename Settings >

```
carl::Bitset smtrat::cad::BaseProjection< Settings >::addPolynomial (
            const Poly & p,
            std::size_t cid,
            bool isBound )  [inline], [inherited]
```

Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.

0.15.378.2.4 addPolynomial() [2/2] template<typename Settings , Backtracking BT>

```
carl::Bitset smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >::addPolynomial (
            const UPoly & p,
            std::size_t cid,
            bool isBound )  [inline], [virtual]
```

Adds the given polynomial to the projection.
Reimplemented from smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >.

0.15.378.2.5 callRemoveCallback() template<typename Settings >

```
void smtrat::cad::BaseProjection< Settings >::callRemoveCallback (
            std::size_t level,
            const SampleLiftedWith & slw ) const  [inline], [protected], [inherited]
```

0.15.378.2.6 canBePurgedByBounds() template<typename Settings >

```
bool smtrat::cad::BaseProjection< Settings >::canBePurgedByBounds (
            const UPoly & p ) const  [inline], [protected], [inherited]
```

Checks whether a polynomial can safely be ignored due to the bounds.

0.15.378.2.7 dim() [1/2] template<typename Settings >

```
std::size_t smtrat::cad::BaseProjection< Settings >::dim ( ) const  [inline], [inherited]
```

Returns the dimension of the projection.

0.15.378.2.8 dim() [2/2] template<typename Settings >

```
std::size_t smtrat::cad::BaseProjection< Settings >::dim  [inline], [inherited]
```

Returns the dimension of the projection.

0.15.378.2.9 empty() [1/2] template<typename Settings >

```
virtual bool smtrat::cad::BaseProjection< Settings >::empty ( )  [inline], [virtual], [inherited]
```

0.15.378.2.10 empty() [2/2] template<typename Settings >

```
bool smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >::empty
(
            std::size_t level ) const  [inline], [override], [virtual], [inherited]
```

Implements smtrat::cad::BaseProjection< Settings >.

0.15.378.2.11 exportAsDot() template<typename Settings >

```
virtual void smtrat::cad::BaseProjection< Settings >::exportAsDot (
            std::ostream &  ) const  [inline], [virtual], [inherited]
```

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, and smtrat::cad::Projection< Incrementality::FULL,

0.15.378.2.12 freeID()

```
template<typename Settings >
void smtrat::cad::BaseProjection< Settings >::freeID (
            std::size_t level,
            std::size_t id )  [inline], [protected], [inherited]
```

Frees a currently used polynomial id for the given level.

0.15.378.2.13 getID()

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::getID (
            std::size_t level )  [inline], [protected], [inherited]
```

Returns a fresh polynomial id for the given level.

0.15.378.2.14 getOrigin()

```
template<typename Settings >
virtual Origin smtrat::cad::BaseProjection< Settings >::getOrigin (
            std::size_t level,
            std::size_t id ) const  [inline], [virtual], [inherited]
```

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, BT, Settings >.

0.15.378.2.15 getPolyForLifting()

```
template<typename Settings >
OptionalID smtrat::cad::BaseProjection< Settings >::getPolyForLifting (
            std::size_t level,
            SampleLiftedWith & slw )  [inline], [inherited]
```

Get a polynomial from this level suited for lifting.

0.15.378.2.16 getPolynomialById()

```
template<typename Settings >
const UPoly& smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings
>::getPolynomialById (
            std::size_t level,
            std::size_t id ) const  [inline], [override], [virtual], [inherited]
```

Retrieves a polynomial from its id.
Implements smtrat::cad::BaseProjection< Settings >.

0.15.378.2.17 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 >.

0.15.378.2.18 isPurged()

```
template<typename Settings >
bool smtrat::cad::BaseProjection< Settings >::isPurged (
            std::size_t level,
            std::size_t id )  [inline], [protected], [inherited]
```

0.15.378.2.19 projectNewPolynomial()

```
template<typename Settings , Backtracking BT>
carl::Bitset smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >::projectNew↩
Polynomial (
            const ConstraintSelection &  = carl::Bitset(true) )  [inline]
```

0.15.378.2.20 removePolynomial() [1/2]

```
template<typename Settings >
void smtrat::cad::BaseProjection< Settings >::removePolynomial (
            const Poly & p,
            std::size_t cid,
            bool isBound )  [inline], [inherited]
```

Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.

0.15.378.2.21 removePolynomial() [2/2]

```
template<typename Settings , Backtracking BT>
void smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >::removePolynomial (
            const UPoly & p,
            std::size_t cid,
            bool isBound )  [inline], [virtual]
```

Removes the given polynomial from the projection.
Reimplemented from smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >.

0.15.378.2.22 reset()

```
template<typename Settings , Backtracking BT>
void smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >::reset ( )  [inline]
```

0.15.378.2.23 setRemoveCallback()

```
template<typename Settings >
template<typename F >
void smtrat::cad::BaseProjection< Settings >::setRemoveCallback (
            F && f )  [inline], [inherited]
```

Sets a callback that is called whenever polynomials are removed.

0.15.378.2.24 size() [1/4]

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::size ( ) const  [inline], [inherited]
```

0.15.378.2.25 size() [2/4]

```
template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::size (
            void  )  [inline], [inherited]
```

0.15.378.2.26 size() [3/4]

```
template<typename Settings >
std::size_t smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings
>::size (
            std::size_t level ) const  [inline], [override], [virtual], [inherited]
```

Implements smtrat::cad::BaseProjection< Settings >.

0.15.378.2.27 size() [4/4]

```
template<typename Settings >
virtual std::size_t smtrat::cad::BaseProjection< Settings >::size (
            void  )  [inherited]
```

0.15.378.2.28 var()

```
template<typename Settings >
carl::Variable smtrat::cad::BaseProjection< Settings >::var (
            std::size_t level ) const  [inline], [protected], [inherited]
```

Returns the variable that corresponds to the given level, that is the variable eliminated in this level.

0.15.378.2.29 vars() template<typename Settings >

```
const auto& smtrat::cad::BaseProjection< Settings >::vars ( ) const [inline], [inherited]
```

Returns the variables used for projection.

0.15.378.3 Friends And Related Function Documentation

0.15.378.3.1 operator<< template<typename Settings , Backtracking BT>

```
template<typename S , Backtracking B>
std::ostream& operator<< (
            std::ostream & os,
            const Projection< Incrementality::SIMPLE, B, S > & p ) [friend]
```

0.15.378.4 Field Documentation

0.15.378.4.1 mConstraints template<typename Settings >

```
const Constraints& smtrat::cad::BaseProjection< Settings >::mConstraints [protected], [inherited]
```

0.15.378.4.2 mInfo template<typename Settings >

```
ProjectionInformation smtrat::cad::BaseProjection< Settings >::mInfo [protected], [inherited]
```

Additional info on projection, projection levels and projection polynomials.

0.15.378.4.3 mRemoveCallback template<typename Settings >

```
std::function<void(std::size_t, const SampleLiftedWith&)> smtrat::cad::BaseProjection< Settings
>::mRemoveCallback [protected], [inherited]
```

Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.

0.15.379 smtrat::cad::projection_compare::ProjectionComparator< Strategy > Struct Template Reference

```
#include <ProjectionComparator.h>
```

0.15.380 smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::D > Struct Reference

```
#include <ProjectionComparator.h>
```

Public Member Functions

- bool operator() (const Candidate< Poly > &lhs, const Candidate< Poly > &rhs) const

0.15.380.1 Member Function Documentation

0.15.380.1.1 operator()() bool smtrat::cad::projection_compare::ProjectionComparator_impl< Args >::operator() (

```
            const Candidate< Poly > & lhs,
            const Candidate< Poly > & rhs ) const [inline], [inherited]
```

0.15.381 smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::ID > Struct Reference

`#include <ProjectionComparator.h>`

Public Member Functions

- bool operator() (const Candidate< Poly > &lhs, const Candidate< Poly > &rhs) const

0.15.381.1 Member Function Documentation

0.15.381.1.1 operator()()

```
bool smtrat::cad::projection_compare::ProjectionComparator_impl< Args >::operator() (
            const Candidate< Poly > & lhs,
            const Candidate< Poly > & rhs ) const  [inline], [inherited]
```

0.15.382 smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::LD > Struct Reference

`#include <ProjectionComparator.h>`

Public Member Functions

- bool operator() (const Candidate< Poly > &lhs, const Candidate< Poly > &rhs) const

0.15.382.1 Member Function Documentation

0.15.382.1.1 operator()()

```
bool smtrat::cad::projection_compare::ProjectionComparator_impl< Args >::operator() (
            const Candidate< Poly > & lhs,
            const Candidate< Poly > & rhs ) const  [inline], [inherited]
```

0.15.383 smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::PD > Struct Reference

`#include <ProjectionComparator.h>`

Public Member Functions

- bool operator() (const Candidate< Poly > &lhs, const Candidate< Poly > &rhs) const

0.15.383.1 Member Function Documentation

0.15.383.1.1 operator()()

```
bool smtrat::cad::projection_compare::ProjectionComparator_impl< Args >::operator() (
            const Candidate< Poly > & lhs,
            const Candidate< Poly > & rhs ) const  [inline], [inherited]
```

0.15.384 smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::SD > Struct Reference

`#include <ProjectionComparator.h>`

Public Member Functions

- bool operator() (const Candidate< Poly > &lhs, const Candidate< Poly > &rhs) const

0.15.384.1 Member Function Documentation

0.15.384.1.1 operator()()

```
bool smtrat::cad::projection_compare::ProjectionComparator_impl< Args >::operator() (
            const Candidate< Poly > & lhs,
            const Candidate< Poly > & rhs ) const  [inline], [inherited]
```

0.15.385 smtrat::cad::projection_compare::ProjectionComparator_impl< Args > Struct Template Reference

```
#include <ProjectionComparator.h>
```

Public Member Functions

- template<typename Poly >
  bool operator() (const Candidate< Poly > &lhs, const Candidate< Poly > &rhs) const

0.15.385.1 Member Function Documentation

0.15.385.1.1 operator()()

```
template<typename...  Args>
template<typename Poly >
bool smtrat::cad::projection_compare::ProjectionComparator_impl< Args >::operator() (
            const Candidate< Poly > & lhs,
            const Candidate< Poly > & rhs ) const  [inline]
```

0.15.386 smtrat::cad::ProjectionGlobalInformation Class Reference

```
#include <ProjectionInformation.h>
```

Data Structures

- struct ECData

Public Types

- using ECMap = std::map< Origin::BaseType, ECData >

Public Member Functions

- void reset (std::size_t dim)
- void createEC (const Origin::BaseType &origin)
- bool isEC (const Origin::BaseType &origin) const
- void addToEC (const Origin::BaseType &origin, std::size_t level, std::size_t pid)
- void removeEC (const Origin::BaseType &origin)

Data Fields

- ECMap mECs
- std::vector< ECMap::const_iterator > mUsedEC
- carl::Bitset mInactive

0.15.386.1 Member Typedef Documentation

0.15.386.1.1 ECMap

```
using smtrat::cad::ProjectionGlobalInformation::ECMap = std::map<Origin::BaseType, ECData>
```

0.15.386.2 Member Function Documentation

0.15.386.2.1 addToEC()

```
void smtrat::cad::ProjectionGlobalInformation::addToEC (
    const Origin::BaseType & origin,
    std::size_t level,
    std::size_t pid ) [inline]
```

0.15.386.2.2 createEC()

```
void smtrat::cad::ProjectionGlobalInformation::createEC (
    const Origin::BaseType & origin ) [inline]
```

0.15.386.2.3 isEC()

```
bool smtrat::cad::ProjectionGlobalInformation::isEC (
    const Origin::BaseType & origin ) const [inline]
```

0.15.386.2.4 removeEC()

```
void smtrat::cad::ProjectionGlobalInformation::removeEC (
    const Origin::BaseType & origin ) [inline]
```

0.15.386.2.5 reset()

```
void smtrat::cad::ProjectionGlobalInformation::reset (
    std::size_t dim ) [inline]
```

0.15.386.3 Field Documentation

0.15.386.3.1 mECs

```
ECMap smtrat::cad::ProjectionGlobalInformation::mECs
```

0.15.386.3.2 mInactive

```
carl::Bitset smtrat::cad::ProjectionGlobalInformation::mInactive
```

0.15.386.3.3 mUsedEC

```
std::vector<ECMap::const_iterator> smtrat::cad::ProjectionGlobal↩
Information::mUsedEC
```

0.15.387 smtrat::cad::ProjectionInformation Class Reference

```
#include <ProjectionInformation.h>
```

Public Member Functions

- const auto & operator() () const
- auto & operator() ()
- const auto & operator() (std::size_t level) const
- auto & operator() (std::size_t level)
- bool hasInfo (std::size_t level) const

- const auto & operator() (std::size_t level, std::size_t pid) const
- auto & operator() (std::size_t level, std::size_t pid)
- void clear (std::size_t level, std::size_t pid)
- void emplace (std::size_t level, std::size_t pid)
- bool hasInfo (std::size_t level, std::size_t pid) const
- bool isActive (std::size_t level, std::size_t id) const
- void reset (std::size_t dim)
- void addECConstraint (std::size_t pid)
- void removeECConstraint (std::size_t)
- bool hasEC (std::size_t level) const
- bool usingEC (std::size_t level) const
- const carl::Bitset & getUsedEC (std::size_t level) const
- bool selectEC (std::size_t level)
- void unselectEC (std::size_t level)

0.15.387.1 Member Function Documentation

0.15.387.1.1 addECConstraint()

```
void smtrat::cad::ProjectionInformation::addECConstraint (
std::size_t pid ) [inline]
```

0.15.387.1.2 clear()

```
void smtrat::cad::ProjectionInformation::clear (
std::size_t level,
std::size_t pid ) [inline]
```

0.15.387.1.3 emplace()

```
void smtrat::cad::ProjectionInformation::emplace (
std::size_t level,
std::size_t pid ) [inline]
```

0.15.387.1.4 getUsedEC()

```
const carl::Bitset& smtrat::cad::ProjectionInformation::getUsedEC (
std::size_t level ) const [inline]
```

0.15.387.1.5 hasEC()

```
bool smtrat::cad::ProjectionInformation::hasEC (
std::size_t level ) const [inline]
```

0.15.387.1.6 hasInfo() [1/2]

```
bool smtrat::cad::ProjectionInformation::hasInfo (
std::size_t level ) const [inline]
```

0.15.387.1.7 hasInfo() [2/2]

```
bool smtrat::cad::ProjectionInformation::hasInfo (
std::size_t level,
std::size_t pid ) const [inline]
```

0.15.387.1.8 isActive()

```
bool smtrat::cad::ProjectionInformation::isActive (
std::size_t level,
std::size_t id ) const [inline]
```

0.15.387.1.9 operator()() [1/6] auto& smtrat::cad::ProjectionInformation::operator() ( ) [inline]

0.15.387.1.10 operator()() [2/6] const auto& smtrat::cad::ProjectionInformation::operator() ( ) const [inline]

0.15.387.1.11 operator()() [3/6] auto& smtrat::cad::ProjectionInformation::operator() (std::size_t *level*) [inline]

0.15.387.1.12 operator()() [4/6] const auto& smtrat::cad::ProjectionInformation::operator() (std::size_t *level*) const [inline]

0.15.387.1.13 operator()() [5/6] auto& smtrat::cad::ProjectionInformation::operator() (std::size_t *level*, std::size_t *pid*) [inline]

0.15.387.1.14 operator()() [6/6] const auto& smtrat::cad::ProjectionInformation::operator() (std::size_t *level*, std::size_t *pid*) const [inline]

0.15.387.1.15 removeECConstraint() void smtrat::cad::ProjectionInformation::removeECConstraint (std::size_t) [inline]

0.15.387.1.16 reset() void smtrat::cad::ProjectionInformation::reset (std::size_t *dim*) [inline]

0.15.387.1.17 selectEC() bool smtrat::cad::ProjectionInformation::selectEC (std::size_t *level*) [inline]

0.15.387.1.18 unselectEC() void smtrat::cad::ProjectionInformation::unselectEC (std::size_t *level*) [inline]

0.15.387.1.19 usingEC() bool smtrat::cad::ProjectionInformation::usingEC (std::size_t *level*) const [inline]

0.15.388 smtrat::cad::ProjectionLevelInformation Class Reference

#include <ProjectionInformation.h>

Data Structures

- struct LevelInfo

Public Member Functions

- bool hasInfo (std::size_t level) const
- void emplace (std::size_t level)
- const auto & operator() (std::size_t level) const
- auto & operator() (std::size_t level)
- void reset (std::size_t dim)

0.15.388.1 Member Function Documentation

0.15.388.1.1 emplace()

```
void smtrat::cad::ProjectionLevelInformation::emplace (
    std::size_t level ) [inline]
```

0.15.388.1.2 hasInfo()

```
bool smtrat::cad::ProjectionLevelInformation::hasInfo (
    std::size_t level ) const [inline]
```

0.15.388.1.3 operator()() [1/2]

```
auto& smtrat::cad::ProjectionLevelInformation::operator() (
    std::size_t level ) [inline]
```

0.15.388.1.4 operator()() [2/2]

```
const auto& smtrat::cad::ProjectionLevelInformation::operator() (
    std::size_t level ) const [inline]
```

0.15.388.1.5 reset()

```
void smtrat::cad::ProjectionLevelInformation::reset (
    std::size_t dim ) [inline]
```

0.15.389 smtrat::cad::ProjectionOperator Struct Reference

```
#include <ProjectionOperator.h>
```

Public Member Functions

- template<typename Callback >
  void operator() (ProjectionType pt, const UPoly &p, carl::Variable variable, Callback &&cb) const
- template<typename Callback >
  void operator() (ProjectionType pt, const UPoly &p, const UPoly &q, carl::Variable variable, Callback &&i) const

0.15.389.1 Member Function Documentation

0.15.389.1.1 operator()() [1/2]

```
template<typename Callback >
void smtrat::cad::ProjectionOperator::operator() (
    ProjectionType pt,
    const UPoly & p,
    carl::Variable variable,
    Callback && cb ) const [inline]
```

0.15.389.1.2 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]
```

0.15.390 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)

0.15.390.1 Member Function Documentation

0.15.390.1.1 clear()

```
void smtrat::cad::ProjectionPolynomialInformation::clear (
            std::size_t level,
            std::size_t pid )  [inline]
```

0.15.390.1.2 emplace()

```
void smtrat::cad::ProjectionPolynomialInformation::emplace (
            std::size_t level,
            std::size_t pid )  [inline]
```

0.15.390.1.3 hasInfo()

```
bool smtrat::cad::ProjectionPolynomialInformation::hasInfo (
            std::size_t level,
            std::size_t pid ) const  [inline]
```

0.15.390.1.4 operator()() [1/2]

```
auto& smtrat::cad::ProjectionPolynomialInformation::operator() (
            std::size_t level,
            std::size_t pid )  [inline]
```

0.15.390.1.5 operator()() [2/2]

```
const auto& smtrat::cad::ProjectionPolynomialInformation::operator()
(
            std::size_t level,
            std::size_t pid ) const  [inline]
```

0.15.390.1.6 reset()

```
void smtrat::cad::ProjectionPolynomialInformation::reset (
            std::size_t  )  [inline]
```

0.15.391 smtrat::cadcells::datastructures::Projections Class Reference

Encapsulates all computations on polynomials.
`#include <projections.h>`

Public Member Functions

- auto main_var (PolyRef p) const
- Projections (PolyPool &pool)
- auto & polys ()
- const auto & polys () const
- void clear_cache (size_t level)

  Clears all polynomials of the specified level and higher in the polynomial cache as well as their projection results.
- void clear_assignment_cache (const Assignment &assignment)

  Clears all projections cached with respect to this assignment.
- PolyRef res (PolyRef p, PolyRef q)
- bool know_disc (PolyRef p) const
- bool known (const Polynomial &p) const
- PolyRef disc (PolyRef p)
- PolyRef ldcf (PolyRef p)
- const std::vector< PolyRef > & factors_nonconst (PolyRef p)
- bool is_zero (const Assignment &sample, PolyRef p)
- size_t num_roots (const Assignment &sample, PolyRef p)
- std::vector< RAN > real_roots (const Assignment &sample, PolyRef p)
- RAN evaluate (const Assignment &sample, IndexedRoot r)
- bool is_nullified (const Assignment &sample, PolyRef p)
- bool is_ldcf_zero (const Assignment &sample, PolyRef p)
- bool is_disc_zero (const Assignment &sample, PolyRef p)
- bool is_const (PolyRef p)
- bool is_zero (PolyRef p)
- bool has_const_coeff (PolyRef p) const
- PolyRef simplest_nonzero_coeff (const Assignment &sample, PolyRef p, std::function< bool(const Polynomial &, const Polynomial &)> compare) const
- std::size_t degree (PolyRef p)
- std::size_t total_degree (PolyRef p)

0.15.391.1 Detailed Description

Encapsulates all computations on polynomials.
Computations are cached with respect to a PolyPool.

0.15.391.2 Constructor & Destructor Documentation

0.15.391.2.1 Projections()

`smtrat::cadcells::datastructures::Projections::Projections ( PolyPool & pool ) [inline]`

0.15.391.3 Member Function Documentation

0.15.391.3.1 clear_assignment_cache()

`void smtrat::cadcells::datastructures::Projections↩ ::clear_assignment_cache ( const Assignment & assignment ) [inline]`

Clears all projections cached with respect to this assignment.

0.15.391.3.2 clear_cache()

```
void smtrat::cadcells::datastructures::Projections::clear_cache (
size_t level ) [inline]
```

Clears all polynomials of the specified level and higher in the polynomial cache as well as their projection results.

0.15.391.3.3 degree()

```
std::size_t smtrat::cadcells::datastructures::Projections::degree (
PolyRef p ) [inline]
```

0.15.391.3.4 disc()

```
PolyRef smtrat::cadcells::datastructures::Projections::disc (
PolyRef p ) [inline]
```

0.15.391.3.5 evaluate()

```
RAN smtrat::cadcells::datastructures::Projections::evaluate (
const Assignment & sample,
IndexedRoot r ) [inline]
```

0.15.391.3.6 factors_nonconst()

```
const std::vector<PolyRef>& smtrat::cadcells::datastructures↩
::Projections::factors_nonconst (
PolyRef p ) [inline]
```

0.15.391.3.7 has_const_coeff()

```
bool smtrat::cadcells::datastructures::Projections::has_const_↩
coeff (
PolyRef p ) const [inline]
```

0.15.391.3.8 is_const()

```
bool smtrat::cadcells::datastructures::Projections::is_const (
PolyRef p ) [inline]
```

0.15.391.3.9 is_disc_zero()

```
bool smtrat::cadcells::datastructures::Projections::is_disc_zero (
const Assignment & sample,
PolyRef p ) [inline]
```

0.15.391.3.10 is_ldcf_zero()

```
bool smtrat::cadcells::datastructures::Projections::is_ldcf_zero (
const Assignment & sample,
PolyRef p ) [inline]
```

0.15.391.3.11 is_nullified()

```
bool smtrat::cadcells::datastructures::Projections::is_nullified (
const Assignment & sample,
PolyRef p ) [inline]
```

0.15.391.3.12 is_zero() [1/2]

```
bool smtrat::cadcells::datastructures::Projections::is_zero (
const Assignment & sample,
PolyRef p ) [inline]
```

0.15.391.3.13 is_zero() [2/2]

```
bool smtrat::cadcells::datastructures::Projections::is_zero (
PolyRef p ) [inline]
```

0.15.391.3.14 know_disc()

```
bool smtrat::cadcells::datastructures::Projections::know_disc (
PolyRef p ) const [inline]
```

0.15.391.3.15 known()

```
bool smtrat::cadcells::datastructures::Projections::known (
const Polynomial & p ) const [inline]
```

0.15.391.3.16 ldcf()

```
PolyRef smtrat::cadcells::datastructures::Projections::ldcf (
PolyRef p ) [inline]
```

0.15.391.3.17 main_var()

```
auto smtrat::cadcells::datastructures::Projections::main_var (
PolyRef p ) const [inline]
```

0.15.391.3.18 num_roots()

```
size_t smtrat::cadcells::datastructures::Projections::num_roots (
const Assignment & sample,
PolyRef p ) [inline]
```

0.15.391.3.19 polys() [1/2]

```
auto& smtrat::cadcells::datastructures::Projections::polys ( )
[inline]
```

0.15.391.3.20 polys() [2/2]

```
const auto& smtrat::cadcells::datastructures::Projections::polys ( )
const [inline]
```

0.15.391.3.21 real_roots()

```
std::vector<RAN> smtrat::cadcells::datastructures::Projections↩
::real_roots (
const Assignment & sample,
PolyRef p ) [inline]
```

0.15.391.3.22 res()

```
PolyRef smtrat::cadcells::datastructures::Projections::res (
PolyRef p,
PolyRef q ) [inline]
```

0.15.391.3.23 simplest_nonzero_coeff()

```
PolyRef smtrat::cadcells::datastructures::Projections↩
::simplest_nonzero_coeff (
const Assignment & sample,
PolyRef p,
std::function< bool(const Polynomial &, const Polynomial &)> compare ) const
[inline]
```

0.15.391.3.24 total_degree()

```
std::size_t smtrat::cadcells::datastructures::Projections::total_↩
degree (
PolyRef p ) [inline]
```

0.15.392 smtrat::cadcells::operators::mccallum_filtered_impl::PropertiesSet Struct Reference

```
#include <operator_mccallum_filtered.h>
```

Public Types

- using type = datastructures::PropertiesT< properties::poly_sgn_inv, properties::poly_irreducible_sgn_inv, properties::poly_semi_sgn_inv, properties::poly_irreducible_semi_sgn_inv, properties::poly_ord_inv, properties::root_well_def, properties::poly_pdel, properties::cell_connected, properties::poly_additional_root_outside, properties::poly_ord_inv_base, properties::root_ordering_holds >

0.15.392.1 Member Typedef Documentation

0.15.392.1.1 type

```
using smtrat::cadcells::operators::mccallum_filtered_impl::PropertiesSet::type
= datastructures::PropertiesT<properties::poly_sgn_inv,properties::poly_irreducible_sgn_inv,properties::poly_
```

0.15.393 smtrat::cadcells::operators::PropertiesSet< Op > Struct Template Reference

0.15.394 smtrat::cadcells::operators::PropertiesSet< op::mccallum > Struct Reference

`#include <operator_mccallum.h>`

Public Types

- using type = datastructures::PropertiesT< properties::poly_sgn_inv, properties::poly_irreducible_sgn_inv, properties::poly_semi_sgn_inv, properties::poly_ord_inv, properties::root_well_def, properties::poly_pdel, properties::cell_connected >

0.15.394.1 Member Typedef Documentation

0.15.394.1.1 type

```
using smtrat::cadcells::operators::PropertiesSet< op::mccallum >::type =
datastructures::PropertiesT<properties::poly_sgn_inv,properties::poly_irreducible_sgn_inv,properties::poly_sem
```

0.15.395 smtrat::cadcells::datastructures::PropertiesT< Ts > Struct Template Reference

Set of properties.
`#include <properties.h>`

0.15.395.1 Detailed Description

template<typename... Ts>
struct smtrat::cadcells::datastructures::PropertiesT< Ts >

Set of properties.
This is a recursive template. The list of template parameters specifies the type of properties which can be hold by this set.
Note that only properties of the same level should be stored within this datastructure.
Properties have the following requirements:

- They need to implement level().
- They need to implement hash_on_level(), operator< and operator== for comparing with properties of the same type and level.
- They need to have a member `static constexpr bool is_flag`. If set to `true`, this indicates that there is only one property of this kind per level and thus, it is not stored in a set but only a Boolean flag is stored.
- They need to implement operator<<.

0.15.396 smtrat::cadcells::datastructures::PropertiesT< T, Ts... > Struct Template Reference

`#include <properties.h>`

Data Fields

- PropertiesTContent< T, T::is_flag >::type content

0.15.396.1 Field Documentation

0.15.396.1.1 content

`template<class T , class... Ts>`
`PropertiesTContent<T, T::is_flag>::type smtrat::cadcells::datastructures::PropertiesT< T, Ts... >::content`

0.15.397 smtrat::cadcells::datastructures::PropertiesTContent< T, is_flag > Struct Template Reference

0.15.398 smtrat::cadcells::datastructures::PropertiesTContent< T, false > Struct Template Reference

`#include <properties.h>`

Public Types

- using type = PropertiesTSet< T >

0.15.398.1 Member Typedef Documentation

0.15.398.1.1 type

`template<typename T >`
`using smtrat::cadcells::datastructures::PropertiesTContent< T, false >::type = PropertiesTSet<T>`

0.15.399 smtrat::cadcells::datastructures::PropertiesTContent< T, true > Struct Template Reference

`#include <properties.h>`

Public Types

- using type = bool

0.15.399.1 Member Typedef Documentation

0.15.399.1.1 type

`template<typename T >`
`using smtrat::cadcells::datastructures::PropertiesTContent< T, true >::type = bool`

0.15.400 smtrat::cadcells::datastructures::property_hash< T > Struct Template Reference

`#include <properties.h>`

Public Member Functions

- std::size_t operator() (const T &p) const

0.15.400.1 Member Function Documentation

0.15.400.1.1 operator()()

```
template<typename T >
std::size_t smtrat::cadcells::datastructures::property_hash< T >::operator() (
            const T & p ) const [inline]
```

0.15.401 smtrat::PseudoBoolEncoder Class Reference

Base class for a PseudoBoolean Encoder.

```
#include <PseudoBoolEncoder.h>
```

Public Member Functions

- std::optional< FormulaT > encode (const ConstraintT &constraint)

  Encodes an arbitrary constraint.
- virtual Rational encodingSize (const ConstraintT &constraint)
- virtual bool canEncode (const ConstraintT &constraint)=0
- virtual std::string name ()

Data Fields

- std::size_t problem_size

Protected Member Functions

- virtual std::optional< FormulaT > doEncode (const ConstraintT &constraint)=0
- FormulaT generateVarChain (const std::set< carl::Variable > &vars, carl::FormulaType type)

0.15.401.1 Detailed Description

Base class for a PseudoBoolean Encoder.
It takes a arithmetic constraint and converts it to a boolean Formula

0.15.401.2 Member Function Documentation

0.15.401.2.1 canEncode()

```
virtual bool smtrat::PseudoBoolEncoder::canEncode (
            const ConstraintT & constraint ) [pure virtual]
```

Implemented in smtrat::TotalizerEncoder, smtrat::ShortFormulaEncoder, smtrat::RNSEncoder, smtrat::MixedSignEncoder, smtrat::LongFormulaEncoder, smtrat::ExactlyOneCommanderEncoder, and smtrat::CardinalityEncoder.

0.15.401.2.2 doEncode()

```
virtual std::optional<FormulaT> smtrat::PseudoBoolEncoder::doEncode (
            const ConstraintT & constraint ) [protected], [pure virtual]
```

Implemented in smtrat::TotalizerEncoder, smtrat::ShortFormulaEncoder, smtrat::RNSEncoder, smtrat::MixedSignEncoder, smtrat::LongFormulaEncoder, smtrat::ExactlyOneCommanderEncoder, and smtrat::CardinalityEncoder.

0.15.401.2.3 encode()

```
std::optional< FormulaT > smtrat::PseudoBoolEncoder::encode (
    const ConstraintT & constraint )
```

Encodes an arbitrary constraint.

Returns

encoded formula

0.15.401.2.4 encodingSize()

```
Rational smtrat::PseudoBoolEncoder::encodingSize (
    const ConstraintT & constraint ) [virtual]
```

Reimplemented in smtrat::TotalizerEncoder, smtrat::ShortFormulaEncoder, smtrat::MixedSignEncoder, smtrat::LongFormulaEncoder, smtrat::ExactlyOneCommanderEncoder, and smtrat::CardinalityEncoder.

0.15.401.2.5 generateVarChain()

```
FormulaT smtrat::PseudoBoolEncoder::generateVarChain (
    const std::set< carl::Variable > & vars,
    carl::FormulaType type ) [protected]
```

0.15.401.2.6 name()

```
virtual std::string smtrat::PseudoBoolEncoder::name ( ) [inline], [virtual]
```

Reimplemented in smtrat::TotalizerEncoder, smtrat::ShortFormulaEncoder, smtrat::MixedSignEncoder, smtrat::LongFormulaEncoder, smtrat::ExactlyOneCommanderEncoder, and smtrat::CardinalityEncoder.

0.15.401.3 Field Documentation

0.15.401.3.1 problem_size

```
std::size_t smtrat::PseudoBoolEncoder::problem_size
```

0.15.402 smtrat::PseudoBoolNormalizer Class Reference

```
#include <PseudoBoolNormalizer.h>
```

Public Member Functions

- std::pair< std::optional< FormulaT >, ConstraintT > normalize (const ConstraintT &constraint)

  Modifies the constraint in-place and returns an additional boolean formula.
- std::map< carl::Variable, carl::Variable > & substitutedVariables ()

  returns all variable substitutions done by this normalizer instance.
- ConstraintT trim (const ConstraintT &constraint)

0.15.402.1 Member Function Documentation

0.15.402.1.1 normalize()

```
std::pair< std::optional< FormulaT >, ConstraintT > smtrat::Pseudo↩
BoolNormalizer::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.

0.15.402.1.2 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.

0.15.402.1.3 trim() `ConstraintT smtrat::PseudoBoolNormalizer::trim ( const ConstraintT & constraint )`

0.15.403 smtrat::parser::QEParser Struct Reference

`#include <Script.h>`

Public Member Functions

- QEParser (Theories *theories)
- carl::Variable resolveVariable (const Identifier &name) const

Data Fields

- Theories * theories
- QualifiedIdentifierParser qualifiedidentifier
- QuantifierParser quantifier
- qi::rule< Iterator, carl::Variable(), Skipper > var
- qi::rule< Iterator, qe::QEQuery(), Skipper > main

0.15.403.1 Constructor & Destructor Documentation

0.15.403.1.1 QEParser() `smtrat::parser::QEParser::QEParser ( Theories * theories ) [inline]`

0.15.403.2 Member Function Documentation

0.15.403.2.1 resolveVariable() `carl::Variable smtrat::parser::QEParser::resolveVariable ( const Identifier & name ) const [inline]`

0.15.403.3 Field Documentation

0.15.403.3.1 main `qi::rule<Iterator, qe::QEQuery(), Skipper> smtrat::parser::QEParser::main`

0.15.403.3.2 qualifiedidentifier `QualifiedIdentifierParser smtrat::parser::QEParser::qualifiedidentifier`

0.15.403.3.3 quantifier `QuantifierParser smtrat::parser::QEParser::quantifier`

0.15.403.3.4 theories `Theories* smtrat::parser::QEParser::theories`

0.15.403.3.5 var `qi::rule<Iterator, carl::Variable(), Skipper> smtrat::parser::QEParser::var`

0.15.404 smtrat::parser::QualifiedIdentifierParser Struct Reference

`#include <Term.h>`

Public Member Functions

- QualifiedIdentifierParser ()
- Identifier checkQualification (const Identifier &identifier, const carl::Sort &) const

Data Fields

- IdentifierParser identifier
- SortParser sort
- qi::rule< Iterator, Identifier(), Skipper > main

0.15.404.1 Constructor & Destructor Documentation

0.15.404.1.1 QualifiedIdentifierParser() `smtrat::parser::QualifiedIdentifierParser::QualifiedIdentifierParser ( ) [inline]`

0.15.404.2 Member Function Documentation

0.15.404.2.1 checkQualification()

```
Identifier smtrat::parser::QualifiedIdentifierParser::checkQualification (
            const Identifier & identifier,
            const carl::Sort & ) const [inline]
```

Todo Check what can be checked here.

0.15.404.3 Field Documentation

0.15.404.3.1 identifier `IdentifierParser smtrat::parser::QualifiedIdentifierParser::identifier`

0.15.404.3.2 main `qi::rule<Iterator, Identifier(), Skipper> smtrat::parser::QualifiedIdentifierParser::main`

0.15.404.3.3 sort `SortParser smtrat::parser::QualifiedIdentifierParser::sort`

0.15.405 smtrat::expression::QuantifierExpression Struct Reference

`#include <ExpressionContent.h>`

Public Member Functions

- QuantifierExpression (QuantifierType _type, std::vector< carl::Variable > &&_variables, Expression &&_expression)

Data Fields

- QuantifierType type
- std::vector< carl::Variable > variables
- Expression expression

0.15.405.1 Constructor & Destructor Documentation

0.15.405.1.1 QuantifierExpression() `smtrat::expression::QuantifierExpression::QuantifierExpression`

```
(
            QuantifierType _type,
            std::vector< carl::Variable > && _variables,
            Expression && _expression )  [inline]
```

0.15.405.2 Field Documentation

0.15.405.2.1 expression `Expression smtrat::expression::QuantifierExpression::expression`

0.15.405.2.3 variables `std::vector<carl::Variable> smtrat::expression::QuantifierExpression↩ ::variables`

0.15.406 smtrat::parser::QuantifierParser Struct Reference

`#include <Script.h>`

Public Member Functions

- QuantifierParser ()

0.15.406.1 Constructor & Destructor Documentation

0.15.406.1.1 QuantifierParser() `smtrat::parser::QuantifierParser::QuantifierParser ( )  [inline]`

0.15.407 Minisat::Queue< T > Class Template Reference

`#include <Queue.h>`

Public Types

- typedef T Key

Public Member Functions

- Queue ()
- void clear (bool dealloc=false)
- int size () const
- const T & operator[ ] (int index) const
- T & operator[ ] (int index)

- T peek () const
- void pop ()
- void insert (T elem)

0.15.407.1 Member Typedef Documentation

0.15.407.1.1 Key `template<class T >`

```
typedef T Minisat::Queue< T >::Key
```

0.15.407.2 Constructor & Destructor Documentation

0.15.407.2.1 Queue() `template<class T >`

```
Minisat::Queue< T >::Queue ( ) [inline]
```

0.15.407.3 Member Function Documentation

0.15.407.3.1 clear() `template<class T >`

```
void Minisat::Queue< T >::clear (
            bool dealloc = false ) [inline]
```

0.15.407.3.2 insert() `template<class T >`

```
void Minisat::Queue< T >::insert (
            T elem ) [inline]
```

0.15.407.3.3 operator[]() [1/2] `template<class T >`

```
T& Minisat::Queue< T >::operator[] (
            int index ) [inline]
```

0.15.407.3.4 operator[]() [2/2] `template<class T >`

```
const T& Minisat::Queue< T >::operator[] (
            int index ) const [inline]
```

0.15.407.3.5 peek() `template<class T >`

```
T Minisat::Queue< T >::peek ( ) const [inline]
```

0.15.407.3.6 pop() `template<class T >`

```
void Minisat::Queue< T >::pop ( ) [inline]
```

0.15.407.3.7 size() `template<class T >`

```
int Minisat::Queue< T >::size ( ) const [inline]
```

0.15.408 smtrat::mcsat::icp::QueueEntry Struct Reference

```
#include <IntervalPropagation.h>
```

Data Fields

- double priority
- carl::contractor::Contractor< FormulaT, Poly > contractor

0.15.408.1 Field Documentation

0.15.408.1.1 contractor `carl::contractor::Contractor<FormulaT, Poly> smtrat::mcsat::icp::↩ QueueEntry::contractor`

0.15.408.1.2 priority `double smtrat::mcsat::icp::QueueEntry::priority`

0.15.409 benchmax::RandomizationAdaptor< T > Class Template Reference

Provides iteration over a given std::vector in a pseudo-randomized order.
`#include <Jobs.h>`

Data Structures

- struct iterator

Public Member Functions

- RandomizationAdaptor (const std::vector< T > &data)

  Construct a randomized view on the given vector.
- auto begin () const

  Return begin of the randomized range.
- auto end () const

  Return end of the randomized range.

0.15.409.1 Detailed Description

template<typename T>
class benchmax::RandomizationAdaptor< T >

Provides iteration over a given std::vector in a pseudo-randomized order.
Internally, it computes a factor that is coprime to the number of jobs (and thus generates all indices). We choose the factor in the order of the square root of the number of jobs.

0.15.409.2 Constructor & Destructor Documentation

0.15.409.2.1 RandomizationAdaptor() `template<typename T >`

```
benchmax::RandomizationAdaptor< T >::RandomizationAdaptor (
            const std::vector< T > & data ) [inline]
```

Construct a randomized view on the given vector.

0.15.409.3 Member Function Documentation

0.15.409.3.1 begin() `template<typename T >`

```
auto benchmax::RandomizationAdaptor< T >::begin ( ) const [inline]
```

Return begin of the randomized range.

0.15.409.3.2 end() `template<typename T >`

`auto benchmax::RandomizationAdaptor< T >::end ( ) const [inline]`

Return end of the randomized range.

0.15.410 smtrat::RationalCapsule Class Reference

`#include <RationalCapsule.h>`

Public Member Functions

- RationalCapsule (const Rational &aRational, const Rational &bRational, const Rational &cRational)
- const Rational & getARational () const
- const Rational & getBRational () const
- const Rational & getCRational () const

Data Fields

- Rational aRational = aRational
- Rational bRational = bRational
- Rational cRational = cRational

0.15.410.1 Constructor & Destructor Documentation

0.15.410.1.1 RationalCapsule()

```
smtrat::RationalCapsule::RationalCapsule (
    const Rational & aRational,
    const Rational & bRational,
    const Rational & cRational )
```

0.15.410.2 Member Function Documentation

0.15.410.2.1 getARational()

`const smtrat::Rational & smtrat::RationalCapsule::getARational ( ) const`

0.15.410.2.2 getBRational()

`const smtrat::Rational & smtrat::RationalCapsule::getBRational ( ) const`

0.15.410.2.3 getCRational()

`const smtrat::Rational & smtrat::RationalCapsule::getCRational ( ) const`

0.15.410.3 Field Documentation

0.15.410.3.1 aRational

`Rational smtrat::RationalCapsule::aRational = aRational`

0.15.410.3.2 bRational

`Rational smtrat::RationalCapsule::bRational = bRational`

0.15.410.3.3 cRational

`Rational smtrat::RationalCapsule::cRational = cRational`

0.15.411 smtrat::parser::RationalPolicies Struct Reference

Specialization of qi::real_policies for a Rational.
`#include <Lexicon.h>`

Static Public Member Functions

- template<typename It , typename Attr >
  static bool parse_nan (It &, It const &, Attr &)
- template<typename It , typename Attr >
  static bool parse_inf (It &, It const &, Attr &)

0.15.411.1 Detailed Description

Specialization of qi::real_policies for a Rational.
Specifies that neither NaN nor Inf is allowed.

0.15.411.2 Member Function Documentation

0.15.411.2.1 parse_inf() template<typename It , typename Attr >

```
static bool smtrat::parser::RationalPolicies::parse_inf (
            It & ,
            It const & ,
            Attr & ) [inline], [static]
```

0.15.411.2.2 parse_nan() template<typename It , typename Attr >

```
static bool smtrat::parser::RationalPolicies::parse_nan (
            It & ,
            It const & ,
            Attr & ) [inline], [static]
```

0.15.412 smtrat::mcsat::onecellcad::RealAlgebraicPoint< Number > Class Template Reference

Represent a multidimensional point whose components are algebraic reals.
`#include <RealAlgebraicPoint.h>`

Public Member Functions

- RealAlgebraicPoint () noexcept=default

  Create an empty point of dimension 0.
- RealAlgebraicPoint (const std::vector< carl::IntRepRealAlgebraicNumber< Number >> &v)

  Convert from a vector using its numbers in the same order as components.
- RealAlgebraicPoint (std::vector< carl::IntRepRealAlgebraicNumber< Number >> &&v)

  Convert from a vector using its numbers in the same order as components.
- RealAlgebraicPoint (const std::list< carl::IntRepRealAlgebraicNumber< Number >> &v)

  Convert from a list using its numbers in the same order as components.
- RealAlgebraicPoint (const std::initializer_list< carl::IntRepRealAlgebraicNumber< Number >> &v)

  Convert from a initializer_list using its numbers in the same order as components.
- std::size_t dim () const

  Give the dimension/number of components of this point.
- RealAlgebraicPoint prefixPoint (size_t componentCount) const

  Make a (lower dimensional) copy that contains only the first 'componentCount'-many components.

- RealAlgebraicPoint conjoin (const carl::IntRepRealAlgebraicNumber< Number > &r)

  Create a new point with another given component added at the end of this point, thereby increasing its dimension by 1.
- const carl::IntRepRealAlgebraicNumber< Number > & operator[ ] (std::size_t index) const

  Retrieve the component of this point at the given index.
- carl::IntRepRealAlgebraicNumber< Number > & operator[ ] (std::size_t index)

  Retrieve the component of this point at the given index.

0.15.412.1 Detailed Description

template<typename Number>
class smtrat::mcsat::onecellcad::RealAlgebraicPoint< Number >

Represent a multidimensional point whose components are algebraic reals.
This class is just a thin wrapper around vector to have a clearer semantic meaning.

0.15.412.2 Constructor & Destructor Documentation

0.15.412.2.1 RealAlgebraicPoint() [1/5] template<typename Number >

```
smtrat::mcsat::onecellcad::RealAlgebraicPoint< Number >::RealAlgebraicPoint ( ) [default],
[noexcept]
```

Create an empty point of dimension 0.

0.15.412.2.2 RealAlgebraicPoint() [2/5] template<typename Number >

```
smtrat::mcsat::onecellcad::RealAlgebraicPoint< Number >::RealAlgebraicPoint (
            const std::vector< carl::IntRepRealAlgebraicNumber< Number >> & v ) [inline],
[explicit]
```

Convert from a vector using its numbers in the same order as components.

0.15.412.2.3 RealAlgebraicPoint() [3/5] template<typename Number >

```
smtrat::mcsat::onecellcad::RealAlgebraicPoint< Number >::RealAlgebraicPoint (
            std::vector< carl::IntRepRealAlgebraicNumber< Number >> && v ) [inline], [explicit]
```

Convert from a vector using its numbers in the same order as components.

0.15.412.2.4 RealAlgebraicPoint() [4/5] template<typename Number >

```
smtrat::mcsat::onecellcad::RealAlgebraicPoint< Number >::RealAlgebraicPoint (
            const std::list< carl::IntRepRealAlgebraicNumber< Number >> & v ) [inline],
[explicit]
```

Convert from a list using its numbers in the same order as components.

0.15.412.2.5 RealAlgebraicPoint() [5/5] template<typename Number >

```
smtrat::mcsat::onecellcad::RealAlgebraicPoint< Number >::RealAlgebraicPoint (
            const std::initializer_list< carl::IntRepRealAlgebraicNumber< Number >> & v )
[inline]
```

Convert from a initializer_list using its numbers in the same order as components.

0.15.412.3 Member Function Documentation

0.15.412.3.1 conjoin() template<typename Number >

```
RealAlgebraicPoint smtrat::mcsat::onecellcad::RealAlgebraicPoint< Number >::conjoin (
            const carl::IntRepRealAlgebraicNumber< Number > & r ) [inline]
```

Create a new point with another given component added at the end of this point, thereby increasing its dimension by 1.
The original point remains untouched.

0.15.412.3.2 dim() template<typename Number >

```
std::size_t smtrat::mcsat::onecellcad::RealAlgebraicPoint< Number >::dim ( ) const [inline]
```

Give the dimension/number of components of this point.

0.15.412.3.3 operator[]() [1/2] template<typename Number >

```
carl::IntRepRealAlgebraicNumber<Number>& smtrat::mcsat::onecellcad::RealAlgebraicPoint< Number
>::operator[] (
            std::size_t index ) [inline]
```

Retrieve the component of this point at the given index.

0.15.412.3.4 operator[]() [2/2] template<typename Number >

```
const carl::IntRepRealAlgebraicNumber<Number>& smtrat::mcsat::onecellcad::RealAlgebraicPoint<
Number >::operator[] (
            std::size_t index ) const [inline]
```

Retrieve the component of this point at the given index.

0.15.412.3.5 prefixPoint() template<typename Number >

```
RealAlgebraicPoint smtrat::mcsat::onecellcad::RealAlgebraicPoint< Number >::prefixPoint (
            size_t componentCount ) const [inline]
```

Make a (lower dimensional) copy that contains only the first 'componentCount'-many components.

0.15.413 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 RealAlgebraicPoint< Rational > &point)
- std::map< carl::Variable, RAN > prefixPointToStdMap (const std::size_t componentCount)

  Create a mapping from the first variables (with index 0 to 'componentCount') in the 'variableOrder' to the first components of the given 'point'.
- std::vector< RAN > isolateLastVariableRoots (const std::size_t polyLevel, const Poly &poly)

  src/lib/datastructures/mcsat/onecellcad/OneCellCAD.h Given a poly p(x1,..,xn), return all roots of the univariate poly p(a1,..,an-1, xn), where a1,..an-1 are the first algebraic real components from 'point' (SORTED!).
- bool vanishesEarly (const std::size_t polyLevel, const Poly &poly)

  Check if an n-variate 'poly' p(x1,..,xn) with n>=1 already becomes the zero poly after plugging in p(a1,..,an-1, xn), where a1,..an-1 are the first n-1 algebraic real components from 'point'.
- bool isPointRootOfPoly (const std::size_t polyLevel, const Poly &poly)
- bool isPointRootOfPoly (const TagPoly &poly)
- std::optional< Poly > coeffNonNull (const TagPoly &boundCandidate)
- bool isMainPointInsideCell (const CADCell &cell)

Data Fields

- const std::vector< carl::Variable > & variableOrder

  Variables can also be indexed by level.
- const RealAlgebraicPoint< Rational > & point

0.15.413.1 Member Function Documentation

0.15.413.1.1 coeffNonNull()

```
std::optional<Poly> smtrat::mcsat::onecellcad::OneCellCAD::coeff↩
NonNull (
            const TagPoly & boundCandidate )  [inline], [inherited]
```

0.15.413.1.2 hasPolyLastVariableRootWithinBounds()

```
bool smtrat::mcsat::onecellcad::recursive↩
::RecursiveCAD::hasPolyLastVariableRootWithinBounds (
            const RAN & low,
            const RAN & high,
            const Poly & poly,
            const std::size_t polyLevel )  [inline]
```

Check if there is a root of the given polynomial—that becomes univariate after pluggin in all but the last variable wrt. the given variableOrder—, that lies between the given 'low' and 'high' bounds.

0.15.413.1.3 isAlreadyProcessed()

```
bool smtrat::mcsat::onecellcad::recursive::RecursiveCAD::is↩
AlreadyProcessed (
            const TagPoly & poly )  [inline]
```

0.15.413.1.4 isMainPointInsideCell()

```
bool smtrat::mcsat::onecellcad::OneCellCAD::isMainPoint↩
InsideCell (
            const CADCell & cell )  [inline], [inherited]
```

0.15.413.1.5 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!).

0.15.413.1.6 isPointRootOfPoly() [1/2]

```
bool smtrat::mcsat::onecellcad::OneCellCAD::isPointRoot↩
OfPoly (
            const std::size_t polyLevel,
            const Poly & poly )  [inline], [inherited]
```

0.15.413.1.7 isPointRootOfPoly() [2/2]

```
bool smtrat::mcsat::onecellcad::OneCellCAD::isPointRoot↩
OfPoly (
            const TagPoly & poly )  [inline], [inherited]
```

0.15.413.1.8 OneCellCAD()

```
smtrat::mcsat::onecellcad::OneCellCAD::OneCellCAD  [inline]
```

0.15.413.1.9 partialDerivativesLayerWithSomeNonEarlyVanishingPoly()

```
std::vector<Poly> smtrat↩
::mcsat::onecellcad::recursive::RecursiveCAD::partialDerivativesLayerWithSomeNonEarlyVanishing↩
Poly (
            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

0.15.413.1.10 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'.

0.15.413.1.11 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'.

0.15.413.1.12 refineNonNull()

```
ShrinkResult smtrat::mcsat::onecellcad::recursive::RecursiveCAD↩
::refineNonNull (
            const TagPoly & boundCandidate,
            CADCell & cell ) [inline]
```

0.15.413.1.13 refineNull()

```
ShrinkResult smtrat::mcsat::onecellcad::recursive::RecursiveCAD↩
::refineNull (
            const TagPoly & boundCandidate,
            CADCell & cell ) [inline]
```

0.15.413.1.14 shrinkCell()

```
ShrinkResult smtrat::mcsat::onecellcad::recursive::RecursiveCAD↩
::shrinkCell (
            const TagPoly & boundCandidate,
            CADCell & cell ) [inline]
```

Merge the given open OpenCADCell 'cell' that contains the given 'point' (called "alpha" in [brown15]) with a polynomial 'poly'.
Before the merge 'cell' represents a region that is sign-invariant on other (previously merged) polynomials (all signs are non-zero). The returned cell represents a region that is additionally sign-invariant on 'poly' (also with non-zero sign).

Returns

either an OpenCADCell or nothing (representing a failed construction)

0.15.413.1.15 shrinkCellWithEarlyVanishingPoly()

```
ShrinkResult smtrat::mcsat::onecellcad::recursive↩
::RecursiveCAD::shrinkCellWithEarlyVanishingPoly (
            const TagPoly & boundCandidate,
            CADCell & cell ) [inline]
```

0.15.413.1.16 shrinkCellWithIrreducibleFactorsOfPoly()

```
ShrinkResult smtrat::mcsat::onecellcad↩
::recursive::RecursiveCAD::shrinkCellWithIrreducibleFactorsOfPoly (
            const TagPoly & poly,
            CADCell & cell ) [inline]
```

0.15.413.1.17 shrinkCellWithNonRootPoint()

```
ShrinkResult smtrat::mcsat::onecellcad::recursive::↩
RecursiveCAD::shrinkCellWithNonRootPoint (
            const TagPoly & boundCandidate,
            CADCell & cell ) [inline]
```

0.15.413.1.18 shrinkCellWithPolyHavingPointAsRoot()

```
ShrinkResult smtrat::mcsat::onecellcad↩
::recursive::RecursiveCAD::shrinkCellWithPolyHavingPointAsRoot (
            const TagPoly & boundCandidate,
            CADCell & cell ) [inline]
```

0.15.413.1.19 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.

0.15.413.1.20 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'.

0.15.413.2 Field Documentation

0.15.413.2.1 point

```
const RealAlgebraicPoint<Rational>& smtrat::mcsat::onecellcad::OneCellCAD↩
::point  [inherited]
```

0.15.413.2.2 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]

0.15.414 smtrat::cad::projection::Reducta< Poly > Struct Template Reference

Construct the set of reducta of the given polynomial.
`#include <utils.h>`

Public Member Functions

- Reducta (const Poly &p)

Data Fields

- T **elements**

  STL member.

0.15.414.1 Detailed Description

template<typename Poly>
struct smtrat::cad::projection::Reducta< Poly >

Construct the set of reducta of the given polynomial.
This only adds a custom constructor to a std::

0.15.414.2 Constructor & Destructor Documentation

0.15.414.2.1 Reducta() `template<typename Poly >`

```
smtrat::cad::projection::Reducta< Poly >::Reducta (
            const Poly & p ) [inline]
```

0.15.414.3 Field Documentation

0.15.414.3.1 elements `T std::vector< T >::elements [inherited]`

STL member.

0.15.415 Minisat::RegionAllocator< T > Class Template Reference

`#include <Alloc.h>`

Public Types

- enum { Ref_Undef = UINT32_MAX }
- enum { Unit_Size = sizeof(uint32_t) }
- typedef uint32_t Ref

Public Member Functions

- RegionAllocator (uint32_t start_cap=1024 ∗1024)
- ∼RegionAllocator ()
- uint32_t size () const
- uint32_t wasted () const
- Ref alloc (int size)
- void free (int size)
- T & operator[ ] (Ref r)
- const T & operator[ ] (Ref r) const
- T ∗ lea (Ref r)
- const T ∗ lea (Ref r) const
- Ref ael (const T ∗t)
- void moveTo (RegionAllocator &to)

0.15.415.1 Member Typedef Documentation

0.15.415.1.1 Ref `template<class T >`

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

0.15.415.2 Member Enumeration Documentation

0.15.415.2.1 anonymous enum `template<class T >`

`anonymous enum`

Enumerator

Ref_Undef	

0.15.415.2.2 anonymous enum `template<class T >`

`anonymous enum`

Enumerator

Unit_Size

0.15.415.3 Constructor & Destructor Documentation

0.15.415.3.1 RegionAllocator() `template<class T >`

```
Minisat::RegionAllocator< T >::RegionAllocator (
            uint32_t start_cap = 1024 * 1024 )  [inline], [explicit]
```

0.15.415.3.2 ~RegionAllocator() `template<class T >`

```
Minisat::RegionAllocator< T >::~RegionAllocator ( )  [inline]
```

0.15.415.4 Member Function Documentation

0.15.415.4.1 ael() `template<class T >`

```
Ref Minisat::RegionAllocator< T >::ael (
            const T * t )  [inline]
```

0.15.415.4.2 alloc() `template<class T >`

```
RegionAllocator< T >::Ref Minisat::RegionAllocator< T >::alloc (
            int size )
```

0.15.415.4.3 free() `template<class T >`

```
void Minisat::RegionAllocator< T >::free (
            int size )  [inline]
```

0.15.415.4.4 lea() [1/2] `template<class T >`

```
T* Minisat::RegionAllocator< T >::lea (
            Ref r )  [inline]
```

0.15.415.4.5 lea() [2/2] `template<class T >`

```
const T* Minisat::RegionAllocator< T >::lea (
            Ref r ) const  [inline]
```

0.15.415.4.6 moveTo() `template<class T >`

```
void Minisat::RegionAllocator< T >::moveTo (
            RegionAllocator< T > & to )  [inline]
```

0.15.415.4.7 operator[]() [1/2] template<class T >

```
T& Minisat::RegionAllocator< T >::operator[] (
            Ref r )  [inline]
```

0.15.415.4.8 operator[]() [2/2] template<class T >

```
const T& Minisat::RegionAllocator< T >::operator[] (
            Ref r ) const  [inline]
```

0.15.415.4.9 size() template<class T >

```
uint32_t Minisat::RegionAllocator< T >::size ( ) const  [inline]
```

0.15.415.4.10 wasted() template<class T >

```
uint32_t Minisat::RegionAllocator< T >::wasted ( ) const  [inline]
```

0.15.416 smtrat::parser::ErrorHandler::result< typename > Struct Template Reference

`#include <Script.h>`

Public Types

- typedef qi::error_handler_result type

0.15.416.1 Member Typedef Documentation

0.15.416.1.1 type template<typename >

```
typedef qi::error_handler_result smtrat::parser::ErrorHandler::result< typename >::type
```

0.15.417 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

0.15.417.1 Constructor & Destructor Documentation

0.15.417.1.1 ResultantRule()

```
smtrat::cad::preprocessor::ResultantRule::ResultantRule (
            Origins & origins,
            const std::vector< carl::Variable > & vars )  [inline]
```

0.15.417.2 Member Function Documentation

0.15.417.2.1 compute()

```
std::optional< std::vector< FormulaT > > smtrat::cad::preprocessor::↩
ResultantRule::compute (
            const std::map< ConstraintT, ConstraintT > & constraints )
```

0.15.417.2.2 getNewECs()

```
const auto& smtrat::cad::preprocessor::ResultantRule::getNewECs ( )
const [inline]
```

0.15.418 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.

0.15.418.1 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.

0.15.418.2 Member Function Documentation

0.15.418.2.1 addResult()

```
void benchmax::Results::addResult (
            const Tool * tool,
            const fs::path & file,
            BenchmarkResult && results ) [inline]
```

Add a new result.

0.15.418.2.2 data()

```
const auto& benchmax::Results::data ( ) const [inline]
```

0.15.418.2.3 get()

```
std::optional<std::reference_wrapper<const BenchmarkResult> > benchmax::↩
Results::get (
            const Tool * tool,
            const fs::path & file ) const [inline]
```

0.15.418.2.4 store() [1/2]

```
void benchmax::Results::store (
            Database & db ) [inline]
```

Store all results to some database.

0.15.418.2.5 store() [2/2]

```
void benchmax::Results::store (
    XMLWriter & xml,
    const Jobs & jobs ) const [inline]
```

Store all results to a xml file.

0.15.419 smtrat::RNSEncoder Class Reference

```
#include <RNSEncoder.h>
```

Public Member Functions

- RNSEncoder ()
- bool canEncode (const ConstraintT &constraint)
- std::optional< FormulaT > encode (const ConstraintT &constraint)

  Encodes an arbitrary constraint.
- virtual Rational encodingSize (const ConstraintT &constraint)
- virtual std::string name ()

Data Fields

- std::size_t problem_size

Protected Member Functions

- std::optional< FormulaT > doEncode (const ConstraintT &constraint)
- FormulaT generateVarChain (const std::set< carl::Variable > &vars, carl::FormulaType type)

0.15.419.1 Constructor & Destructor Documentation

0.15.419.1.1 RNSEncoder()

```
smtrat::RNSEncoder::RNSEncoder ( ) [inline]
```

0.15.419.2 Member Function Documentation

0.15.419.2.1 canEncode()

```
bool smtrat::RNSEncoder::canEncode (
    const ConstraintT & constraint ) [virtual]
```

Implements smtrat::PseudoBoolEncoder.

0.15.419.2.2 doEncode()

```
std::optional< FormulaT > smtrat::RNSEncoder::doEncode (
    const ConstraintT & constraint ) [protected], [virtual]
```

Implements smtrat::PseudoBoolEncoder.

0.15.419.2.3 encode()

```
std::optional< FormulaT > smtrat::PseudoBoolEncoder::encode (
    const ConstraintT & constraint ) [inherited]
```

Encodes an arbitrary constraint.

Returns

encoded formula

0.15.419.2.4 encodingSize() `Rational smtrat::PseudoBoolEncoder::encodingSize ( const ConstraintT & constraint ) [virtual], [inherited]`

Reimplemented in smtrat::TotalizerEncoder, smtrat::ShortFormulaEncoder, smtrat::MixedSignEncoder, smtrat::LongFormulaEncoder, smtrat::ExactlyOneCommanderEncoder, and smtrat::CardinalityEncoder.

0.15.419.2.5 generateVarChain() `FormulaT smtrat::PseudoBoolEncoder::generateVarChain ( const std::set< carl::Variable > & vars, carl::FormulaType type ) [protected], [inherited]`

0.15.419.2.6 name() `virtual std::string smtrat::PseudoBoolEncoder::name ( ) [inline], [virtual], [inherited]`

Reimplemented in smtrat::TotalizerEncoder, smtrat::ShortFormulaEncoder, smtrat::MixedSignEncoder, smtrat::LongFormulaEncoder, smtrat::ExactlyOneCommanderEncoder, and smtrat::CardinalityEncoder.

0.15.419.3 Field Documentation

0.15.419.3.1 problem_size `std::size_t smtrat::PseudoBoolEncoder::problem_size [inherited]`

0.15.420 smtrat::cadcells::operators::properties::root_ordering_holds Struct Reference

`#include <properties.h>`

Public Member Functions

- size_t level () const
- std::size_t hash_on_level () const

Data Fields

- datastructures::IndexedRootOrdering ordering
- std::size_t lvl

Static Public Attributes

- static constexpr bool is_flag = false

0.15.420.1 Member Function Documentation

0.15.420.1.1 hash_on_level() `std::size_t smtrat::cadcells::operators::properties::root_ordering↩_holds::hash_on_level ( ) const [inline]`

0.15.420.1.2 level() `size_t smtrat::cadcells::operators::properties::root_ordering_holds::level ( ) const [inline]`

0.15.420.2 Field Documentation

0.15.420.2.1 is_flag `constexpr bool smtrat::cadcells::operators::properties::root_ordering_↩holds::is_flag = false [static], [constexpr]`

0.15.420.2.2 lvl `std::size_t smtrat::cadcells::operators::properties::root_ordering_holds::lvl`

0.15.420.2.3 ordering `datastructures::IndexedRootOrdering smtrat::cadcells::operators::properties↩ ::root_ordering_holds::ordering`

0.15.421 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

0.15.421.1 Member Function Documentation

0.15.421.1.1 hash_on_level() `std::size_t smtrat::cadcells::operators::properties::root_well_↩ def::hash_on_level ( ) const [inline]`

0.15.421.1.2 level() `size_t smtrat::cadcells::operators::properties::root_well_def::level ( ) const [inline]`

0.15.421.2 Field Documentation

0.15.421.2.1 is_flag `constexpr bool smtrat::cadcells::operators::properties::root_well_def::is↩ _flag = false [static], [constexpr]`

0.15.421.2.2 root `datastructures::IndexedRoot smtrat::cadcells::operators::properties::root_↩ well_def::root`

0.15.422 smtrat::cadcells::datastructures::RootFunction Class Reference

`#include <roots.h>`

Public Member Functions

- RootFunction (IndexedRoot data)
- RootFunction (CompoundMinMax &&data)
- RootFunction (CompoundMaxMin &&data)
- bool is_root () const
- bool is_cminmax () const
- bool is_cmaxmin () const

- const IndexedRoot & root () const
- const CompoundMinMax & cminmax () const
- const CompoundMaxMin & cmaxmin () const
- const auto & roots () const
- void polys (boost::container::flat_set< PolyRef > &result) const
- boost::container::flat_set< PolyRef > polys () const
- bool has_poly (const PolyRef poly) const
- std::optional< IndexedRoot > poly_root_below (const PolyRef poly) const
- std::optional< IndexedRoot > poly_root_above (const PolyRef poly) const

Friends

- bool operator== (const RootFunction &lhs, const RootFunction &rhs)
- bool operator< (const RootFunction &lhs, const RootFunction &rhs)
- std::ostream & operator<< (std::ostream &os, const RootFunction &data)

0.15.422.1 Constructor & Destructor Documentation

0.15.422.1.1 RootFunction() [1/3]

`smtrat::cadcells::datastructures::RootFunction::RootFunction ( IndexedRoot data ) [inline]`

0.15.422.1.2 RootFunction() [2/3]

`smtrat::cadcells::datastructures::RootFunction::RootFunction ( CompoundMinMax && data ) [inline]`

0.15.422.1.3 RootFunction() [3/3]

`smtrat::cadcells::datastructures::RootFunction::RootFunction ( CompoundMaxMin && data ) [inline]`

0.15.422.2 Member Function Documentation

0.15.422.2.1 cmaxmin()

`const CompoundMaxMin& smtrat::cadcells::datastructures::RootFunction↩ ::cmaxmin ( ) const [inline]`

0.15.422.2.2 cminmax()

`const CompoundMinMax& smtrat::cadcells::datastructures::RootFunction↩ ::cminmax ( ) const [inline]`

0.15.422.2.3 has_poly()

`bool smtrat::cadcells::datastructures::RootFunction::has_poly ( const PolyRef poly ) const [inline]`

0.15.422.2.4 is_cmaxmin()

`bool smtrat::cadcells::datastructures::RootFunction::is_cmaxmin ( ) const [inline]`

0.15.422.2.5 is_cminmax()

`bool smtrat::cadcells::datastructures::RootFunction::is_cminmax ( ) const [inline]`

0.15.422.2.6 is_root()

`bool smtrat::cadcells::datastructures::RootFunction::is_root ( ) const [inline]`

0.15.422.2.7 poly_root_above()

```
std::optional<IndexedRoot> smtrat::cadcells::datastructures::↩
RootFunction::poly_root_above (
            const PolyRef poly ) const  [inline]
```

0.15.422.2.8 poly_root_below()

```
std::optional<IndexedRoot> smtrat::cadcells::datastructures::↩
RootFunction::poly_root_below (
            const PolyRef poly ) const  [inline]
```

0.15.422.2.9 polys() [1/2]

`boost::container::flat_set<PolyRef> smtrat::cadcells::datastructures↩ ::RootFunction::polys ( ) const [inline]`

0.15.422.2.10 polys() [2/2]

```
void smtrat::cadcells::datastructures::RootFunction::polys (
            boost::container::flat_set< PolyRef > & result ) const  [inline]
```

0.15.422.2.11 root()

`const IndexedRoot& smtrat::cadcells::datastructures::RootFunction::root ( ) const [inline]`

0.15.422.2.12 roots()

`const auto& smtrat::cadcells::datastructures::RootFunction::roots ( ) const [inline]`

0.15.422.3 Friends And Related Function Documentation

0.15.422.3.1 operator<

```
bool operator< (
            const RootFunction & lhs,
            const RootFunction & rhs )  [friend]
```

0.15.422.3.2 operator<<

```
std::ostream& operator<< (
            std::ostream & os,
            const RootFunction & data )  [friend]
```

0.15.422.3.3 operator==

```
bool operator== (
            const RootFunction & lhs,
            const RootFunction & rhs )  [friend]
```

0.15.423 smtrat::mcsat::arithmetic::RootIndexer< RANT > Class Template Reference

`#include <RootIndexer.h>`

Public Member Functions

- void add (const std::vector< RANT > &list)
- void process ()
- std::size_t size () const

- std::size_t operator[ ] (const RANT &ran) const
- const RANT & operator[ ] (std::size_t n) const
- const RANT & sampleFrom (std::size_t n) const
- bool is_root (std::size_t n) const

0.15.423.1 Member Function Documentation

0.15.423.1.1 add()

```
template<typename RANT >
void smtrat::mcsat::arithmetic::RootIndexer< RANT >::add (
            const std::vector< RANT > & list ) [inline]
```

0.15.423.1.2 is_root()

```
template<typename RANT >
bool smtrat::mcsat::arithmetic::RootIndexer< RANT >::is_root (
            std::size_t n ) const [inline]
```

0.15.423.1.3 operator[]() [1/2]

```
template<typename RANT >
std::size_t smtrat::mcsat::arithmetic::RootIndexer< RANT >::operator[] (
            const RANT & ran ) const [inline]
```

0.15.423.1.4 operator[]() [2/2]

```
template<typename RANT >
const RANT& smtrat::mcsat::arithmetic::RootIndexer< RANT >::operator[] (
            std::size_t n ) const [inline]
```

0.15.423.1.5 process()

```
template<typename RANT >
void smtrat::mcsat::arithmetic::RootIndexer< RANT >::process ( ) [inline]
```

0.15.423.1.6 sampleFrom()

```
template<typename RANT >
const RANT& smtrat::mcsat::arithmetic::RootIndexer< RANT >::sampleFrom (
            std::size_t n ) const [inline]
```

0.15.423.1.7 size()

```
template<typename RANT >
std::size_t smtrat::mcsat::arithmetic::RootIndexer< RANT >::size ( ) const [inline]
```

0.15.424 smtrat::cad::Sample Class Reference

```
#include <Sample.h>
```

Public Member Functions

- Sample (const RAN &value)
- Sample (const RAN &value, bool isRoot)
- Sample (const RAN &value, std::size_t id)
- const RAN & value () const
- bool isRoot () const
- void setIsRoot (bool isRoot)
- const SampleLiftedWith & liftedWith () const
- SampleLiftedWith & liftedWith ()
- const SampleRootOf & rootOf () const

- SampleRootOf & rootOf ()
- const carl::Bitset & evaluatedWith () const
- carl::Bitset & evaluatedWith ()
- const carl::Bitset & evaluationResult () const
- carl::Bitset & evaluationResult ()
- auto getConflictingConstraints () const
- bool hasConflictWithConstraint () const
- void merge (const Sample &s)
- bool operator< (const Sample &s) const
- bool operator> (const Sample &s) const
- bool operator== (const Sample &s) const

Friends

- std::ostream & operator<< (std::ostream &os, const Sample &s)

0.15.424.1 Constructor & Destructor Documentation

0.15.424.1.1 Sample() [1/3]

```
smtrat::cad::Sample::Sample (
    const RAN & value ) [inline], [explicit]
```

0.15.424.1.2 Sample() [2/3]

```
smtrat::cad::Sample::Sample (
    const RAN & value,
    bool isRoot ) [inline], [explicit]
```

0.15.424.1.3 Sample() [3/3]

```
smtrat::cad::Sample::Sample (
    const RAN & value,
    std::size_t id ) [inline], [explicit]
```

0.15.424.2 Member Function Documentation

0.15.424.2.1 evaluatedWith() [1/2]

```
carl::Bitset& smtrat::cad::Sample::evaluatedWith ( ) [inline]
```

0.15.424.2.2 evaluatedWith() [2/2]

```
const carl::Bitset& smtrat::cad::Sample::evaluatedWith ( )
const [inline]
```

0.15.424.2.3 evaluationResult() [1/2]

```
carl::Bitset& smtrat::cad::Sample::evaluationResult ( )
[inline]
```

0.15.424.2.4 evaluationResult() [2/2]

```
const carl::Bitset& smtrat::cad::Sample::evaluationResult (
) const [inline]
```

0.15.424.2.5 getConflictingConstraints()

```
auto smtrat::cad::Sample::getConflictingConstraints ( )
const [inline]
```

0.15.424.2.6 hasConflictWithConstraint() `bool smtrat::cad::Sample::hasConflictWithConstraint ( ) const [inline]`

0.15.424.2.7 isRoot() `bool smtrat::cad::Sample::isRoot ( ) const [inline]`

0.15.424.2.8 liftedWith() [1/2] `SampleLiftedWith& smtrat::cad::Sample::liftedWith ( ) [inline]`

0.15.424.2.9 liftedWith() [2/2] `const SampleLiftedWith& smtrat::cad::Sample::liftedWith ( ) const [inline]`

0.15.424.2.10 merge()

```
void smtrat::cad::Sample::merge (
    const Sample & s ) [inline]
```

0.15.424.2.11 operator<()

```
bool smtrat::cad::Sample::operator< (
    const Sample & s ) const [inline]
```

0.15.424.2.12 operator==()

```
bool smtrat::cad::Sample::operator== (
    const Sample & s ) const [inline]
```

0.15.424.2.13 operator>()

```
bool smtrat::cad::Sample::operator> (
    const Sample & s ) const [inline]
```

0.15.424.2.14 rootOf() [1/2] `SampleRootOf& smtrat::cad::Sample::rootOf ( ) [inline]`

0.15.424.2.15 rootOf() [2/2] `const SampleRootOf& smtrat::cad::Sample::rootOf ( ) const [inline]`

0.15.424.2.16 setIsRoot()

```
void smtrat::cad::Sample::setIsRoot (
    bool isRoot ) [inline]
```

0.15.424.2.17 value() `const RAN& smtrat::cad::Sample::value ( ) const [inline]`

0.15.424.3 Friends And Related Function Documentation

0.15.424.3.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const Sample & s ) [friend]
```

0.15.425 smtrat::cad::sample_compare::SampleComparator< Iterator, Strategy > Struct Template Reference

`#include <SampleComparator.h>`

0.15.426 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LS > Struct Template Reference

`#include <SampleComparator.h>`

Public Member Functions

- bool operator() (const Iterator &lhs, const Iterator &rhs) const

0.15.426.1 Member Function Documentation

0.15.426.1.1 operator()() `bool smtrat::cad::sample_compare::SampleComparator_impl< Iterator , Args >::operator() (`
`const Iterator & lhs,`
`const Iterator & rhs ) const [inline], [inherited]`

0.15.427 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LT > Struct Template Reference

`#include <SampleComparator.h>`

Public Member Functions

- bool operator() (const Iterator &lhs, const Iterator &rhs) const

0.15.427.1 Member Function Documentation

0.15.427.1.1 operator()() `bool smtrat::cad::sample_compare::SampleComparator_impl< Iterator , Args >::operator() (`
`const Iterator & lhs,`
`const Iterator & rhs ) const [inline], [inherited]`

0.15.428 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LTA > Struct Template Reference

`#include <SampleComparator.h>`

Public Member Functions

- bool operator() (const Iterator &lhs, const Iterator &rhs) const

0.15.428.1 Member Function Documentation

0.15.428.1.1 operator()() `bool smtrat::cad::sample_compare::SampleComparator_impl< Iterator , Args >::operator() (`
`const Iterator & lhs,`
`const Iterator & rhs ) const [inline], [inherited]`

0.15.429 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LTS > Struct Template Reference

`#include <SampleComparator.h>`

Public Member Functions

- bool operator() (const Iterator &lhs, const Iterator &rhs) const

0.15.429.1 Member Function Documentation

0.15.429.1.1 operator()()

```
bool smtrat::cad::sample_compare::SampleComparator_impl< Iterator ,
Args >::operator() (
            const Iterator & lhs,
            const Iterator & rhs ) const  [inline], [inherited]
```

0.15.430 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LTSA > Struct Template Reference

```
#include <SampleComparator.h>
```

Public Member Functions

- bool operator() (const Iterator &lhs, const Iterator &rhs) const

0.15.430.1 Member Function Documentation

0.15.430.1.1 operator()()

```
bool smtrat::cad::sample_compare::SampleComparator_impl< Iterator ,
Args >::operator() (
            const Iterator & lhs,
            const Iterator & rhs ) const  [inline], [inherited]
```

0.15.431 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::S > Struct Template Reference

```
#include <SampleComparator.h>
```

Public Member Functions

- bool operator() (const Iterator &lhs, const Iterator &rhs) const

0.15.431.1 Member Function Documentation

0.15.431.1.1 operator()()

```
bool smtrat::cad::sample_compare::SampleComparator_impl< Iterator ,
Args >::operator() (
            const It & lhs,
            const It & rhs ) const  [inline], [inherited]
```

0.15.432 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::T > Struct Template Reference

```
#include <SampleComparator.h>
```

Public Member Functions

- bool operator() (const Iterator &lhs, const Iterator &rhs) const

0.15.432.1 Member Function Documentation

0.15.432.1.1 operator()()

```
bool smtrat::cad::sample_compare::SampleComparator_impl< Iterator , Args >::operator() (
            const Iterator & lhs,
            const Iterator & rhs ) const  [inline], [inherited]
```

0.15.433 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::TLSA > Struct Template Reference

```
#include <SampleComparator.h>
```

Public Member Functions

- bool operator() (const Iterator &lhs, const Iterator &rhs) const

0.15.433.1 Member Function Documentation

0.15.433.1.1 operator()()

```
bool smtrat::cad::sample_compare::SampleComparator_impl< Iterator , Args >::operator() (
            const Iterator & lhs,
            const Iterator & rhs ) const  [inline], [inherited]
```

0.15.434 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::TS > Struct Template Reference

```
#include <SampleComparator.h>
```

Public Member Functions

- bool operator() (const Iterator &lhs, const Iterator &rhs) const

0.15.434.1 Member Function Documentation

0.15.434.1.1 operator()()

```
bool smtrat::cad::sample_compare::SampleComparator_impl< Iterator , Args >::operator() (
            const Iterator & lhs,
            const Iterator & rhs ) const  [inline], [inherited]
```

0.15.435 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::TSA > Struct Template Reference

```
#include <SampleComparator.h>
```

Public Member Functions

- bool operator() (const Iterator &lhs, const Iterator &rhs) const

0.15.435.1 Member Function Documentation

0.15.435.1.1 operator()()

```
bool smtrat::cad::sample_compare::SampleComparator_impl< Iterator ,
Args >::operator() (
            const Iterator & lhs,
            const Iterator & rhs ) const  [inline], [inherited]
```

0.15.436 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Type > Struct Template Reference

```
#include <SampleComparator.h>
```

Public Member Functions

- bool operator() (const Iterator &lhs, const Iterator &rhs) const
- bool reference (const Iterator &lhs, const Iterator &rhs) const
- bool compare (const Iterator &lhs, const Iterator &rhs) const

0.15.436.1 Member Function Documentation

0.15.436.1.1 compare()

```
template<typename Iterator >
bool smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Type >↩
::compare (
            const Iterator & lhs,
            const Iterator & rhs ) const  [inline]
```

0.15.436.1.2 operator()()

```
template<typename Iterator >
bool smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Type >↩
::operator() (
            const Iterator & lhs,
            const Iterator & rhs ) const  [inline]
```

0.15.436.1.3 reference()

```
template<typename Iterator >
bool smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Type >↩
::reference (
            const Iterator & lhs,
            const Iterator & rhs ) const  [inline]
```

0.15.437 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Value > Struct Template Reference

```
#include <SampleComparator.h>
```

Public Member Functions

- bool operator() (const Iterator &lhs, const Iterator &rhs) const

0.15.437.1 Member Function Documentation

0.15.437.1.1 operator()()

```
bool smtrat::cad::sample_compare::SampleComparator_impl< Iterator ,
Args >::operator() (
            const Iterator & lhs,
            const Iterator & rhs ) const  [inline], [inherited]
```

0.15.438 smtrat::cad::sample_compare::SampleComparator_impl< It, Args > Struct Template Reference

`#include <SampleComparator.h>`

Public Member Functions

- bool operator() (const It &lhs, const It &rhs) const

0.15.438.1 Member Function Documentation

0.15.438.1.1 operator()()

```
template<typename It , typename... Args>
bool smtrat::cad::sample_compare::SampleComparator_impl< It, Args >::operator() (
            const It & lhs,
            const It & rhs ) const [inline]
```

0.15.439 smtrat::cadcells::datastructures::SampledDerivation< Properties > Class Template Reference

A SampledDerivation is a DelineatedDerivation with a sample and an DelineationInterval w.r.t.

`#include <derivation.h>`

Public Member Functions

- SampledDerivation (DelineatedDerivationRef< Properties > base, RAN main_sample)
- DelineatedDerivationRef< Properties > & delineated ()
- const DelineatedDerivationRef< Properties > & delineated () const
- const DelineationInterval & cell () const
- void delineate_cell ()

  Determines the cell w.r.t.
- const Assignment & sample () const
- const RAN & main_var_sample () const
- BaseDerivationRef< Properties > & base ()
- const BaseDerivationRef< Properties > & base () const
- Delineation & delin ()
- const Assignment & underlying_sample () const
- DerivationRef< Properties > & underlying ()
- PolyPool & polys ()
- Projections & proj ()
- carl::Variable main_var () const
- size_t level () const
- template<typename P >
  void insert (P property)
- template<typename P >
  bool contains (const P &property) const
- template<typename P >
  const PropertiesTSet< P > & properties () const
- void merge_with (const SampledDerivation< Properties > &other)

0.15.439.1 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

0.15.439.2 Constructor & Destructor Documentation

0.15.439.2.1 SampledDerivation()

```
template<typename Properties >
smtrat::cadcells::datastructures::SampledDerivation< Properties >::SampledDerivation (
            DelineatedDerivationRef< Properties > base,
            RAN main_sample ) [inline]
```

0.15.439.3 Member Function Documentation

0.15.439.3.1 base() [1/2]

```
template<typename Properties >
BaseDerivationRef<Properties>& smtrat::cadcells::datastructures::SampledDerivation< Properties
>::base ( ) [inline]
```

0.15.439.3.2 base() [2/2]

```
template<typename Properties >
const BaseDerivationRef<Properties>& smtrat::cadcells::datastructures::SampledDerivation<
Properties >::base ( ) const [inline]
```

0.15.439.3.3 cell()

```
template<typename Properties >
const DelineationInterval& smtrat::cadcells::datastructures::SampledDerivation< Properties >↩
::cell ( ) const [inline]
```

0.15.439.3.4 contains()

```
template<typename Properties >
template<typename P >
bool smtrat::cadcells::datastructures::SampledDerivation< Properties >::contains (
            const P & property ) const [inline]
```

0.15.439.3.5 delin()

```
template<typename Properties >
Delineation& smtrat::cadcells::datastructures::SampledDerivation< Properties >::delin ( )
[inline]
```

0.15.439.3.6 delineate_cell()

```
template<typename Properties >
void smtrat::cadcells::datastructures::SampledDerivation< Properties >::delineate_cell ( )
[inline]
```

Determines the cell w.r.t.
the delineation.

0.15.439.3.7 delineated() [1/2]

```
template<typename Properties >
DelineatedDerivationRef<Properties>& smtrat::cadcells::datastructures::SampledDerivation<
Properties >::delineated ( ) [inline]
```

0.15.439.3.8 delineated() [2/2]

template<typename Properties >
const DelineatedDerivationRef<Properties>& smtrat::cadcells::datastructures::SampledDerivation< Properties >::delineated ( ) const [inline]

0.15.439.3.9 insert()

template<typename Properties >
template<typename P >
void smtrat::cadcells::datastructures::SampledDerivation< Properties >::insert (P *property*) [inline]

0.15.439.3.10 level()

template<typename Properties >
size_t smtrat::cadcells::datastructures::SampledDerivation< Properties >::level ( ) const [inline]

0.15.439.3.11 main_var()

template<typename Properties >
carl::Variable smtrat::cadcells::datastructures::SampledDerivation< Properties >::main_var ( ) const [inline]

0.15.439.3.12 main_var_sample()

template<typename Properties >
const RAN& smtrat::cadcells::datastructures::SampledDerivation< Properties >::main_var_sample ( ) const [inline]

0.15.439.3.13 merge_with()

template<typename Properties >
void smtrat::cadcells::datastructures::SampledDerivation< Properties >::merge_with (const SampledDerivation< Properties > & *other*) [inline]

0.15.439.3.14 polys()

template<typename Properties >
PolyPool& smtrat::cadcells::datastructures::SampledDerivation< Properties >::polys ( ) [inline]

0.15.439.3.15 proj()

template<typename Properties >
Projections& smtrat::cadcells::datastructures::SampledDerivation< Properties >::proj ( ) [inline]

0.15.439.3.16 properties()

template<typename Properties >
template<typename P >
const PropertiesTSet<P>& smtrat::cadcells::datastructures::SampledDerivation< Properties >↩
::properties ( ) const [inline]

0.15.439.3.17 sample()

template<typename Properties >
const Assignment& smtrat::cadcells::datastructures::SampledDerivation< Properties >::sample ( ) const [inline]

0.15.439.3.18 underlying()

template<typename Properties >
DerivationRef<Properties>& smtrat::cadcells::datastructures::SampledDerivation< Properties >::underlying ( ) [inline]

0.15.439.3.19 underlying_sample()

```
template<typename Properties >
const Assignment& smtrat::cadcells::datastructures::SampledDerivation< Properties >::underlying↩
_sample ( ) const [inline]
```

0.15.440 smtrat::SampledDerivationRefCompare Struct Reference

```
#include <Helper.h>
```

Public Member Functions

- template<typename T >
  constexpr bool operator() (const cadcells::datastructures::SampledDerivationRef< T > &a, const cadcells::datastructures::SampledDerivationRef< T > &b) const

0.15.440.1 Member Function Documentation

0.15.440.1.1 operator()()

```
template<typename T >
constexpr bool smtrat::SampledDerivationRefCompare::operator() (
            const cadcells::datastructures::SampledDerivationRef< T > & a,
            const cadcells::datastructures::SampledDerivationRef< T > & b ) const [inline],
[constexpr]
```

0.15.441 smtrat::cad::SampleIteratorQueue< Iterator, Comparator > Class Template Reference

```
#include <SampleIteratorQueue.h>
```

Public Member Functions

- auto begin () const
- auto end () const
- template<typename InputIt >
  void assign (InputIt begin, InputIt end)
- void clear ()
- template<typename Filter >
  void cleanup (Filter &&f)
- bool empty () const
- Iterator getNextSample ()
- void addNewSample (Iterator it)
- template<typename InputIt >
  void addNewSamples (InputIt begin, InputIt end)
- Iterator removeNextSample ()
- bool is_consistent () const

0.15.441.1 Member Function Documentation

0.15.441.1.1 addNewSample()

```
template<typename Iterator , typename Comparator >
void smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::addNewSample (
            Iterator it ) [inline]
```

0.15.441.1.2 addNewSamples()

```
template<typename Iterator , typename Comparator >
template<typename InputIt >
void smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::addNewSamples (
            InputIt begin,
            InputIt end )  [inline]
```

0.15.441.1.3 assign()

```
template<typename Iterator , typename Comparator >
template<typename InputIt >
void smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::assign (
            InputIt begin,
            InputIt end )  [inline]
```

0.15.441.1.4 begin()

```
template<typename Iterator , typename Comparator >
auto smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::begin ( ) const  [inline]
```

0.15.441.1.5 cleanup()

```
template<typename Iterator , typename Comparator >
template<typename Filter >
void smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::cleanup (
            Filter && f )  [inline]
```

0.15.441.1.6 clear()

```
template<typename Iterator , typename Comparator >
void smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::clear ( )  [inline]
```

0.15.441.1.7 empty()

```
template<typename Iterator , typename Comparator >
bool smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::empty ( ) const  [inline]
```

0.15.441.1.8 end()

```
template<typename Iterator , typename Comparator >
auto smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::end ( ) const  [inline]
```

0.15.441.1.9 getNextSample()

```
template<typename Iterator , typename Comparator >
Iterator smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::getNextSample ( )  [inline]
```

0.15.441.1.10 is_consistent()

```
template<typename Iterator , typename Comparator >
bool smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::is_consistent ( ) const  [inline]
```

0.15.441.1.11 removeNextSample()

```
template<typename Iterator , typename Comparator >
Iterator smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::removeNextSample ( )  [inline]
```

0.15.442 smtrat::covering_ng::sampling< S > Struct Template Reference

```
#include <Sampling.h>
```

Static Public Member Functions

- template<typename FE , cadcells::operators::op op>
  static std::optional< cadcells::RAN > sample_outside (const IntervalSet< op > &derivations, const FE &f)

0.15.442.1 Member Function Documentation

0.15.442.1.1 sample_outside()

```
template<SamplingAlgorithm S>
template<typename FE , cadcells::operators::op op>
static std::optional<cadcells::RAN> smtrat::covering_ng::sampling< S >::sample_outside (
            const IntervalSet< op > & derivations,
            const FE & f )  [static]
```

0.15.443 smtrat::sampling< S > Struct Template Reference

```
#include <Sampling.h>
```

Static Public Member Functions

- template<typename T >
  static size_t sample_outside (cadcells::RAN &sample, const std::set< cadcells::datastructures::SampledDerivationRef< T >, SampledDerivationRefCompare > &derivations)

0.15.443.1 Member Function Documentation

0.15.443.1.1 sample_outside()

```
template<SamplingAlgorithm S>
template<typename T >
static size_t smtrat::sampling< S >::sample_outside (
            cadcells::RAN & sample,
            const std::set< cadcells::datastructures::SampledDerivationRef< T >, SampledDerivationRefCompare
> & derivations )  [static]
```

0.15.444 smtrat::covering_ng::sampling< SamplingAlgorithm::LOWER_UPPER_BETWEEN_SAMPLING > Struct Reference

```
#include <Sampling.h>
```

Static Public Member Functions

- template<typename FE , cadcells::operators::op op>
  static std::optional< cadcells::RAN > sample_outside (const IntervalSet< op > &intervals, const FE &)

0.15.444.1 Member Function Documentation

0.15.444.1.1 sample_outside()

```
template<typename FE , cadcells::operators::op op>
static std::optional<cadcells::RAN> smtrat::covering_ng::sampling< SamplingAlgorithm::LOWER_UPPER_BETWEEN_SAM
>::sample_outside (
            const IntervalSet< op > & intervals,
            const FE &  )  [inline], [static]
```

0.15.445 smtrat::sampling< SamplingAlgorithm::LOWER_UPPER_BETWEEN_SAMPLING > Struct Reference

```
#include <Sampling.h>
```

Static Public Member Functions

- template<typename T >
  static size_t sample_outside (cadcells::RAN &sample, const std::set< cadcells::datastructures::SampledDerivationRef< T >, SampledDerivationRefCompare > &derivationsSet)

0.15.445.1 Member Function Documentation

0.15.445.1.1 sample_outside() template<typename T >

```
static size_t smtrat::sampling< SamplingAlgorithm::LOWER_UPPER_BETWEEN_SAMPLING >::sample_↩
outside (
            cadcells::RAN & sample,
            const std::set< cadcells::datastructures::SampledDerivationRef< T >, SampledDerivationRefCompare
> & derivationsSet )  [inline], [static]
```

0.15.446 smtrat::SATModule< Settings > Class Template Reference

Implements a module performing DPLL style SAT checking.

```
#include <SATModule.h>
```

Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- SATModule (const ModuleInput *_formula, Conditionals &_foundAnswer, Manager *const _manager=nullptr)

  Constructs a SATModule.
- ~SATModule ()

  Destructs this SATModule.
- bool addCore (ModuleInput::const_iterator)

  The module has to take the given sub-formula of the received formula into account.
- Answer checkCore ()

  Checks the received formula for consistency.
- void removeCore (ModuleInput::const_iterator)

  Removes everything related to the given sub-formula of the received formula.
- void updateModel () const

  Updates the model, if the solver has detected the consistency of the received formula, beforehand.
- void updateAllModels ()

  Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
- void updateInfeasibleSubset ()

  Updates the infeasible subset found by the SATModule, if the received formula is unsatisfiable.
- void cleanUpAfterOptimizing (const std::vector< Minisat::CRef > &_excludedAssignments)
- void removeUpperBoundOnMinimal ()
- void addConstraintToInform (const FormulaT &)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- void addConstraintToInform_ (const FormulaT &_formula)
- void addBooleanAssignments (RationalAssignment &_rationalAssignment) const

  Adds the Boolean assignments to the given assignments, which were determined by the Minisat procedure.
- void print (std::ostream &_out=std::cout, const std::string &_init="") const

  Prints everything.
- void printCurrentAssignment (std::ostream &=std::cout, const std::string &="") 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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] 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.

- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

  Copies the infeasible subsets of the passed formula.
- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())

- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.

- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

Friends

- class VarSchedulerBase
- class VarSchedulerMcsatBase

0.15.446.1 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.

0.15.446.2 Member Typedef Documentation

0.15.446.2.1 SettingsType `template<class Settings >`

```
typedef Settings smtrat::SATModule< Settings >::SettingsType
```

0.15.446.3 Member Enumeration Documentation

0.15.446.3.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

0.15.446.4 Constructor & Destructor Documentation

0.15.446.4.1 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.

0.15.446.4.2 ~SATModule() `template<class Settings >`

`smtrat::SATModule< Settings >::~SATModule ( )`

Destructs this SATModule.

0.15.446.5 Member Function Documentation

0.15.446.5.1 add() `bool smtrat::Module::add (`

`ModuleInput::const_iterator _subformula ) [inherited]`

The module has to take the given sub-formula of the received formula into account.

Parameters

_subformula	The sub-formula to take additionally into account.

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

0.15.446.5.2 addBooleanAssignments() `template<class Settings >`

`void smtrat::SATModule< Settings >::addBooleanAssignments (`

`RationalAssignment & _rationalAssignment ) const`

Adds the Boolean assignments to the given assignments, which were determined by the Minisat procedure.
Note: Assignments in the given map are not overwritten.

Parameters

_rationalAssignment	The assignments to add the Boolean assignments to.

0.15.446.5.3 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.

0.15.446.5.4 addConstraintToInform_() `template<class Settings >`

`void smtrat::SATModule< Settings >::addConstraintToInform_ (`

`const FormulaT & _formula ) [inline]`

0.15.446.5.5 addCore() [1/2] `virtual bool smtrat::Module::addCore (`

`[[maybe_unused] ] 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.

0.15.446.5.6 addCore() [2/2] template<class Settings >

```
bool smtrat::SATModule< Settings >::addCore (
            ModuleInput::const_iterator  )
```

The module has to take the given sub-formula of the received formula into account.

Parameters

_subformula	The sub-formula to take additionally into account.

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

0.15.446.5.7 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula
(
            const FormulaT & formula )  [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.446.5.8 addLemma()

```
void smtrat::Module::addLemma (
            const FormulaT & _lemma,
            const LemmaType & _lt = LemmaType::NORMAL,
            const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) )  [inline],
[inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.446.5.9 addOrigin()

```
void smtrat::Module::addOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.446.5.10 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.446.5.11 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.446.5.12 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.446.5.13 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.

0.15.446.5.14 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns

All satisfying assignments, if existent.

0.15.446.5.15 anAnswerFound()

```
bool smtrat::Module::anAnswerFound ( ) const [inline], [protected],
[inherited]
```

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.446.5.16 answerFound()

```
const smtrat::Conditionals& smtrat::Module::answerFound ( ) const
[inline], [inherited]
```

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.446.5.17 backendsModel()

```
const Model & smtrat::Module::backendsModel ( ) const [protected],
[inherited]
```

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.446.5.18 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.446.5.19 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.446.5.20 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.446.5.21 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.446.5.22 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.

0.15.446.5.23 checkCore()

```
template<class Settings >
Answer smtrat::SATModule< Settings >::checkCore ( ) [virtual]
```

Checks the received formula for consistency.

Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.446.5.24 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.

0.15.446.5.25 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.

0.15.446.5.26 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.446.5.27 cleanUpAfterOptimizing() `template<class Settings >`

```
void smtrat::SATModule< Settings >::cleanUpAfterOptimizing (
            const std::vector< Minisat::CRef > & _excludedAssignments )
```

0.15.446.5.28 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

0.15.446.5.29 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

0.15.446.5.30 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

0.15.446.5.31 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.446.5.32 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
            const FormulaT & _formula,
            FormulaSetT & _origins ) const [inherited]
```

0.15.446.5.33 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.

0.15.446.5.34 collectStats() `template<class Settings >`

```
void smtrat::SATModule< Settings >::collectStats ( )
```

Collects the taken statistics.

0.15.446.5.35 collectTheoryPropagations()

`void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

0.15.446.5.36 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ToInform ( ) const [inline], [inherited]`

Returns

The constraints which the backends must be informed about.

0.15.446.5.37 currentlySatisfied()

`virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.446.5.38 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)

0.15.446.5.39 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.

0.15.446.5.40 deinformCore()

`virtual void smtrat::Module::deinformCore ( [[maybe_unused] ] 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.

0.15.446.5.41 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

0.15.446.5.42 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 >.

0.15.446.5.43 excludeNotReceivedVariablesFromModel()

`void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

0.15.446.5.44 findBestOrigin()

`std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin ( const std::vector< FormulaT > & _origins ) const [protected], [inherited]`

Parameters

_origins	

Returns

0.15.446.5.45 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.

0.15.446.5.46 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.

0.15.446.5.47 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.446.5.48 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.446.5.49 getBackendsAllModels()

`void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.446.5.50 getBackendsModel()

`void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.446.5.51 getInfeasibleSubsets() [1/2]

`void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.446.5.52 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.

0.15.446.5.53 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.446.5.54 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.

0.15.446.5.55 getOrigins() [1/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.446.5.56 getOrigins() [2/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.446.5.57 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.

0.15.446.5.58 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.

0.15.446.5.59 hasLemmas() `bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

0.15.446.5.60 hasValidInfeasibleSubset() `bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]`

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.446.5.61 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.446.5.62 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.

0.15.446.5.63 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.

0.15.446.5.64 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.

0.15.446.5.65 informCore()

`virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.446.5.66 informedConstraints()

`const carl::FastSet<FormulaT>& smtrat::Module::informedConstraints ( ) const [inline], [inherited]`

Returns

The position of the first constraint of which no backend has been informed about.

0.15.446.5.67 init()

`void smtrat::Module::init ( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not be called twice and only before the first runBackends call.
Reimplemented in smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntBlastModule< Settings >, smtrat::FPPModule< Settings >, smtrat::CurryModule< Settings >, and smtrat::BVModule< Settings >.

0.15.446.5.68 is_minimizing()

`bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.446.5.69 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.

0.15.446.5.70 isPreprocessor()

`bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.446.5.71 lemmas()

`const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.446.5.72 merge()

```
std::vector< FormulaT > smtrat::Module::merge (
    const std::vector< FormulaT > & _vecSetA,
    const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]
```

Merges the two vectors of sets into the first one.
({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

Parameters

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns

The vector being the two given vectors merged.

0.15.446.5.73 model()

`const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.446.5.74 modelsDisjoint()

```
bool smtrat::Module::modelsDisjoint (
    const Model & _modelA,
    const Model & _modelB ) [static], [protected], [inherited]
```

Checks whether there is no variable assigned by both models.

Parameters

_modelA	The first model to check for.
_modelB	The second model to check for.

Returns

true, if there is no variable assigned by both models; false, otherwise.

0.15.446.5.75 moduleName()

`virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]`

Returns

The name of the given module type as name.

Reimplemented in smtrat::VSModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::STropModule< Settings >, smtrat::SplitSOSModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntEqModule< Settings >, smtrat::IntBlastModule< Settings >, smtrat::IncWidthModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::GBModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::CurryModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::BVModule< Settings >, and smtrat::BEModule< Settings >.

0.15.446.5.76 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

0.15.446.5.77 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.446.5.78 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.446.5.79 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.446.5.80 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.446.5.81 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.446.5.82 print() [1/2]

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.446.5.83 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.

0.15.446.5.84 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.

0.15.446.5.85 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.

0.15.446.5.86 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.

0.15.446.5.87 printClause() [1/2] template<class Settings >

```
void smtrat::SATModule< Settings >::printClause (
            const Minisat::vec< Minisat::Lit > & ,
            bool _withAssignment = false,
            std::ostream & _out = std::cout,
            const std::string & _init = "" ) const
```

Prints the clause resulting from the given vector of literals.

Parameters

_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.

0.15.446.5.88 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.

0.15.446.5.89 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.

0.15.446.5.90 printClauses() template<class Settings >

```
void smtrat::SATModule< Settings >::printClauses (
            const Minisat::vec< Minisat::CRef > & _clauses,
            const std::string _name,
            std::ostream & _out = std::cout,
            const std::string & _init = "",
            int  = 0,
            bool _withAssignment = false,
            bool _onlyNotSatisfied = false ) const
```

Prints all given clauses.

Parameters

_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.

0.15.446.5.91 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.

0.15.446.5.92 printCurrentAssignment()

```
template<class Settings >
void smtrat::SATModule< Settings >::printCurrentAssignment (
            std::ostream &  = std::cout,
            const std::string &  = "" ) const
```

Prints the current assignment of the SAT solver.

Parameters

_out	The output stream where the answer should be printed.
_init	The line initiation.

0.15.446.5.93 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.

0.15.446.5.94 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.

0.15.446.5.95 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.446.5.96 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.

0.15.446.5.97 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.

0.15.446.5.98 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.446.5.99 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.

0.15.446.5.100 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.446.5.101 probablyLooping()

```
bool smtrat::Module::probablyLooping (
const typename Poly::PolyType & _branchingPolynomial,
const Rational & _branchingValue ) const [protected], [inherited]
```

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.446.5.102 receivedFormulaChecked()

```
void smtrat::Module::receivedFormulaChecked ( ) [inline],
[inherited]
```

Notifies that the received formulas has been checked.

0.15.446.5.103 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩
InfeasibleSubset (
ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

0.15.446.5.104 receivedVariable()

```
bool smtrat::Module::receivedVariable (
carl::Variable::Arg _var ) const [inline], [inherited]
```

0.15.446.5.105 remove()

```
void smtrat::Module::remove (
ModuleInput::const_iterator _subformula ) [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

Parameters

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

0.15.446.5.106 removeCore() [1/2]

```
virtual void smtrat::Module::removeCore (
            [[maybe_unused] ] 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.

0.15.446.5.107 removeCore() [2/2]

```
template<class Settings >
void smtrat::SATModule< Settings >::removeCore (
            ModuleInput::const_iterator  )
```

Removes everything related to the given sub-formula of the received formula.

Parameters

_subformula	The sub formula of the received formula to remove.

0.15.446.5.108 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origin (
            ModuleInput::iterator _formula,
            const FormulaT & _origin )  [inline], [protected], [inherited]
```

0.15.446.5.109 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins )  [inline], [protected],
[inherited]
```

0.15.446.5.110 removeUpperBoundOnMinimal()

```
template<class Settings >
void smtrat::SATModule< Settings >::removeUpperBoundOnMinimal ( )
```

0.15.446.5.111 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.446.5.112 rReceivedFormula() `const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]`

Returns

A reference to the formula passed to this module.

0.15.446.5.113 runBackends() [1/2] `virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]`

Reimplemented in smtrat::PModule.

0.15.446.5.114 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.

0.15.446.5.115 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.

0.15.446.5.116 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.

0.15.446.5.117 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.

0.15.446.5.118 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 !=.

0.15.446.5.119 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.446.5.120 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.

0.15.446.5.121 updateInfeasibleSubset() `template<class Settings > void smtrat::SATModule< Settings >::updateInfeasibleSubset ( )`

Updates the infeasible subset found by the SATModule, if the received formula is unsatisfiable.

0.15.446.5.122 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.446.5.123 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.

0.15.446.5.124 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).

0.15.446.6 Friends And Related Function Documentation

0.15.446.6.1 VarSchedulerBase `template<class Settings >` `friend class VarSchedulerBase [friend]`

0.15.446.6.2 VarSchedulerMcsatBase `template<class Settings >` `friend class VarSchedulerMcsatBase [friend]`

0.15.446.7 Field Documentation

0.15.446.7.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.446.7.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.446.7.3 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.

0.15.446.7.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.446.7.5 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

0.15.446.7.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.446.7.7 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.

0.15.446.7.8 mFirstUncheckedReceivedSubformula

`ModuleInput::const_iterator smtrat::Module↩::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.446.7.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.446.7.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.446.7.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.446.7.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.446.7.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std↩::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.446.7.14 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.446.7.15 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.446.7.16 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.446.7.17 mObjectiveVariable

`carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.446.7.18 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplitting↩Variables [static], [inherited]`

Reusable splitting variables.

0.15.446.7.19 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.446.7.20 mSmallerMusesCheckCounter

`unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.446.7.21 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.

0.15.446.7.22 mTheoryPropagations

`std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

0.15.446.7.23 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).

0.15.446.7.24 mVariableCounters

`std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.447 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

0.15.447.1 Constructor & Destructor Documentation

0.15.447.1.1 ScriptParser() template<typename Callee >

```
smtrat::parser::ScriptParser< Callee >::ScriptParser (
            InstructionHandler & h,
            Theories & theories,
            Callee & callee )  [inline]
```

0.15.447.2 Member Function Documentation

0.15.447.2.1 startFunctionDefinition() template<typename Callee >

```
void smtrat::parser::ScriptParser< Callee >::startFunctionDefinition ( )  [inline]
```

0.15.447.3 Field Documentation

0.15.447.3.1 attribute template<typename Callee >

```
AttributeParser smtrat::parser::ScriptParser< Callee >::attribute
```

0.15.447.3.2 callee template<typename Callee >

```
Callee& smtrat::parser::ScriptParser< Callee >::callee
```

0.15.447.3.3 command template<typename Callee >

```
qi::rule<Iterator, Skipper> smtrat::parser::ScriptParser< Callee >::command
```

0.15.447.3.4 decimal template<typename Callee >

```
DecimalParser smtrat::parser::ScriptParser< Callee >::decimal
```

0.15.447.3.5 errorHandler template<typename Callee >

```
px::function<ErrorHandler> smtrat::parser::ScriptParser< Callee >::errorHandler
```

0.15.447.3.6 functionDefinition template<typename Callee >

```
qi::rule<Iterator, Skipper> smtrat::parser::ScriptParser< Callee >::functionDefinition
```

0.15.447.3.7 functionDefinitionArg template<typename Callee >

```
qi::rule<Iterator, types::VariableType(), Skipper> smtrat::parser::ScriptParser< Callee >↩
::functionDefinitionArg
```

0.15.447.3.8 handler template<typename Callee >

```
InstructionHandler& smtrat::parser::ScriptParser< Callee >::handler
```

0.15.447.3.9 keyword template<typename Callee >

```
KeywordParser smtrat::parser::ScriptParser< Callee >::keyword
```

0.15.447.3.10 logic template<typename Callee >

LogicParser smtrat::parser::ScriptParser< Callee >::logic

0.15.447.3.11 main template<typename Callee >

qi::rule<Iterator, Skipper> smtrat::parser::ScriptParser< Callee >::main

0.15.447.3.12 numeral template<typename Callee >

NumeralParser smtrat::parser::ScriptParser< Callee >::numeral

0.15.447.3.13 qeQuery template<typename Callee >

QEParser smtrat::parser::ScriptParser< Callee >::qeQuery

0.15.447.3.14 sort template<typename Callee >

SortParser smtrat::parser::ScriptParser< Callee >::sort

0.15.447.3.15 sortedvariable template<typename Callee >

SortedVariableParser smtrat::parser::ScriptParser< Callee >::sortedvariable

0.15.447.3.16 state template<typename Callee >

ParserState smtrat::parser::ScriptParser< Callee >::state

0.15.447.3.17 string template<typename Callee >

StringParser smtrat::parser::ScriptParser< Callee >::string

0.15.447.3.18 symbol template<typename Callee >

SymbolParser smtrat::parser::ScriptParser< Callee >::symbol

0.15.447.3.19 term template<typename Callee >

TermParser smtrat::parser::ScriptParser< Callee >::term

0.15.447.3.20 theories template<typename Callee >

Theories& smtrat::parser::ScriptParser< Callee >::theories

0.15.448 smtrat::parser::ParserState::ScriptScope Struct Reference

#include <ParserState.h>

Public Member Functions

- ScriptScope (const ParserState &state)
- void discharge (ParserState &state)

0.15.448.1 Constructor & Destructor Documentation

0.15.448.1.1 ScriptScope() `smtrat::parser::ParserState::ScriptScope::ScriptScope ( const ParserState & state ) [inline]`

0.15.448.2 Member Function Documentation

0.15.448.2.1 discharge() `void smtrat::parser::ParserState::ScriptScope::discharge ( ParserState & state ) [inline]`

0.15.449 smtrat::mcsat::onecellcad::Section Struct Reference

Represent a cell's (closed-interval-boundary) component along th k-th axis.
`#include <utils.h>`

Data Fields

- MultivariateRootT boundFunction
- RAN isolatedRoot

  For performance we cache the isolated root from 'boundFunction' that lies closest along this section's level to the main point.

0.15.449.1 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.

0.15.449.2 Field Documentation

0.15.449.2.1 boundFunction `MultivariateRootT smtrat::mcsat::onecellcad::Section::boundFunction`

0.15.449.2.2 isolatedRoot `RAN smtrat::mcsat::onecellcad::Section::isolatedRoot`

For performance we cache the isolated root from 'boundFunction' that lies closest along this section's level to the main point.

0.15.450 smtrat::onecellcad::recursive::Section Struct Reference

Represent a cell's closed-interval-boundary object along one single axis by an irreducible, multivariate polynomial of level k.
`#include <OpenCAD.h>`

Data Fields

- MultiPoly poly
- RAN cachedPoint

  A single, special bound after having plugged a specific point of level k-1 can be cached for performance (needed for [1]).

0.15.450.1 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).

0.15.450.2 Field Documentation

0.15.450.2.1 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]).

0.15.450.2.2 poly `MultiPoly smtrat::onecellcad::recursive::Section::poly`

0.15.451 smtrat::mcsat::onecellcad::levelwise::SectionHeuristic1 Struct Reference

`#include <Explanation.h>`

Static Public Attributes

- static constexpr int sectionHeuristic = 1

0.15.451.1 Field Documentation

0.15.451.1.1 sectionHeuristic `constexpr int smtrat::mcsat::onecellcad::levelwise::Section↩ Heuristic1::sectionHeuristic = 1 [static], [constexpr]`

0.15.452 smtrat::mcsat::onecellcad::levelwise::SectionHeuristic2 Struct Reference

`#include <Explanation.h>`

Static Public Attributes

- static constexpr int sectionHeuristic = 2

0.15.452.1 Field Documentation

0.15.452.1.1 sectionHeuristic `constexpr int smtrat::mcsat::onecellcad::levelwise::Section↩ Heuristic2::sectionHeuristic = 2 [static], [constexpr]`

0.15.453 smtrat::mcsat::onecellcad::levelwise::SectionHeuristic3 Struct Reference

`#include <Explanation.h>`

Static Public Attributes

- static constexpr int sectionHeuristic = 3

0.15.453.1 Field Documentation

0.15.453.1.1 sectionHeuristic `constexpr int smtrat::mcsat::onecellcad::levelwise::Section↩ Heuristic3::sectionHeuristic = 3 [static], [constexpr]`

0.15.454 smtrat::mcsat::onecellcad::Sector Struct Reference

Represent a cell's open-interval boundary object along one single axis by two irreducible, multivariate polynomials of level k.

`#include <utils.h>`

Data Fields

- std::optional< Section > lowBound

  A std::nullopt lowBound represents negative infinity.
- std::optional< Section > highBound

  A std:nullopt highBound represents infinity.

0.15.454.1 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.

0.15.454.2 Field Documentation

0.15.454.2.1 highBound `std::optional<Section> smtrat::mcsat::onecellcad::Sector::highBound`

A std:nullopt highBound represents infinity.

0.15.454.2.2 lowBound `std::optional<Section> smtrat::mcsat::onecellcad::Sector::lowBound`

A std::nullopt lowBound represents negative infinity.

0.15.455 smtrat::onecellcad::recursive::Sector Struct Reference

Represent a cell's open-interval boundary object along one single axis by two irreducible, multivariate polynomials of level k.

`#include <OpenCAD.h>`

Public Member Functions

- bool isLowBoundNegInfty () const
- bool isHighBoundInfty () const

Data Fields

- std::optional< Section > lowBound
- std::optional< Section > highBound

0.15.455.1 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.

0.15.455.2 Member Function Documentation

0.15.455.2.1 isHighBoundInfty() `bool smtrat::onecellcad::recursive::Sector::isHighBoundInfty ( ) const [inline]`

0.15.455.2.2 isLowBoundNegInfty() `bool smtrat::onecellcad::recursive::Sector::isLowBoundNeg↩ Infty ( ) const [inline]`

0.15.455.3 Field Documentation

0.15.455.3.1 highBound `std::optional<Section> smtrat::onecellcad::recursive::Sector::high↩ Bound`

0.15.455.3.2 lowBound `std::optional<Section> smtrat::onecellcad::recursive::Sector::lowBound`

0.15.456 smtrat::mcsat::onecellcad::levelwise::SectorHeuristic1 Struct Reference

`#include <Explanation.h>`

Static Public Attributes

- static constexpr int sectorHeuristic = 1

0.15.456.1 Field Documentation

0.15.456.1.1 sectorHeuristic `constexpr int smtrat::mcsat::onecellcad::levelwise::SectorHeuristic1↩ ::sectorHeuristic = 1 [static], [constexpr]`

0.15.457 smtrat::mcsat::onecellcad::levelwise::SectorHeuristic2 Struct Reference

`#include <Explanation.h>`

Static Public Attributes

- static constexpr int sectorHeuristic = 2

0.15.457.1 Field Documentation

0.15.457.1.1 sectorHeuristic `constexpr int smtrat::mcsat::onecellcad::levelwise::SectorHeuristic2↩ ::sectorHeuristic = 2 [static], [constexpr]`

0.15.458 smtrat::mcsat::onecellcad::levelwise::SectorHeuristic3 Struct Reference

`#include <Explanation.h>`

Static Public Attributes

- static constexpr int sectorHeuristic = 3

0.15.458.1 Field Documentation

0.15.458.1.1 sectorHeuristic `constexpr int smtrat::mcsat::onecellcad::levelwise::SectorHeuristic3↩ ::sectorHeuristic = 3 [static], [constexpr]`

0.15.459 smtrat::subtropical::Separator Struct Reference

Represents the class of all original constraints with the same left hand side after a normalization.
`#include <Subtropical.h>`

Public Member Functions

- Separator (const Poly &normalization)

Data Fields

- const carl::Variable bias

  Bias variable of the separating hyperplane.
- const std::vector< Vertex > vertices

  Vertices for all terms of the normalized left hand side.

0.15.459.1 Detailed Description

Represents the class of all original constraints with the same left hand side after a normalization.
Here, the set of all received relations of constraints with the same left hand side is stored. At any one time only one relation can be active and used for linearization.

0.15.459.2 Constructor & Destructor Documentation

0.15.459.2.1 Separator() `smtrat::subtropical::Separator::Separator ( const Poly & normalization ) [inline]`

0.15.459.3 Field Documentation

0.15.459.3.1 bias `const carl::Variable smtrat::subtropical::Separator::bias`

Bias variable of the separating hyperplane.

0.15.459.3.2 vertices `const std::vector<Vertex> smtrat::subtropical::Separator::vertices`

Vertices for all terms of the normalized left hand side.

0.15.460 smtrat::mcsat::SequentialAssignment< Backends > Struct Template Reference

`#include <SequentialAssignment.h>`

0.15.461 smtrat::mcsat::SequentialExplanation< Backends > Struct Template Reference

`#include <SequentialExplanation.h>`

0.15.462 benchmax::settings::Settings Struct Reference

Generic class to manage runtime settings.
`#include <Settings.h>`

0.15.462.1 Detailed Description

Generic class to manage runtime settings.
Essentially stores all (named) settings in a map<string,any> and retrieves settings classes when requested.

0.15.463 smtrat::settings::Settings Struct Reference

`#include <Settings.h>`

0.15.464 smtrat::SettingsComponents Class Reference

`#include <SettingsComponents.h>`

Public Member Functions

- void add (std::function< void(SettingsParser &)> &&f)
- void add_to_parser (SettingsParser &parser) const

0.15.464.1 Member Function Documentation

0.15.464.1.1 add()

```
void smtrat::SettingsComponents::add (
    std::function< void(SettingsParser &)> && f ) [inline]
```

0.15.464.1.2 add_to_parser()

```
void smtrat::SettingsComponents::add_to_parser (
    SettingsParser & parser ) const [inline]
```

0.15.465 benchmax::SettingsParser Class Reference

Generic class to manage settings parsing using boost::program_options.
`#include <SettingsParser.h>`

0.15.465.1 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.

0.15.466 smtrat::SettingsParser Class Reference

`#include <SettingsParser.h>`

0.15.467 smtrat::CNFerModule::SettingsType Struct Reference

`#include <CNFerModule.h>`

Static Public Attributes

- static constexpr auto moduleName = "CNFerModule"

0.15.467.1 Field Documentation

0.15.467.1.1 moduleName `constexpr auto smtrat::CNFerModule::SettingsType::moduleName = "CNFer↩ Module" [static], [constexpr]`

0.15.468 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

0.15.468.1 Constructor & Destructor Documentation

0.15.468.1.1 SExpressionParser() `smtrat::parser::SExpressionParser::SExpressionParser ( ) [inline]`

0.15.468.2 Field Documentation

0.15.468.2.1 keyword `KeywordParser smtrat::parser::SExpressionParser::keyword`

0.15.468.2.2 main `qi::rule<Iterator, SExpression<types::ConstType>), Skipper> smtrat::parser↩ ::SExpressionParser::main`

0.15.468.2.3 specconstant `SpecConstantParser smtrat::parser::SExpressionParser::specconstant`

0.15.468.2.4 symbol `SymbolParser smtrat::parser::SExpressionParser::symbol`

0.15.469 smtrat::parser::SExpressionSequence< T > Struct Template Reference

`#include <Common.h>`

Public Member Functions

- SExpressionSequence (const std::vector< SExpression< T >> &v)
- SExpressionSequence (std::vector< SExpression< T >> &&v)

Data Fields

- T **elements**

  STL member.

0.15.469.1 Constructor & Destructor Documentation

0.15.469.1.1 SExpressionSequence() [1/2]

```
template<typename T >
smtrat::parser::SExpressionSequence< T >::SExpressionSequence (
            const std::vector< SExpression< T >> & v ) [inline]
```

0.15.469.1.2 SExpressionSequence() [2/2]

```
template<typename T >
smtrat::parser::SExpressionSequence< T >::SExpressionSequence (
            std::vector< SExpression< T >> && v ) [inline]
```

0.15.469.2 Field Documentation

0.15.469.2.1 elements

```
T std::vector< T >::elements [inherited]
```

STL member.

0.15.470 smtrat::ShortFormulaEncoder Class Reference

```
#include <ShortFormulaEncoder.h>
```

Public Member Functions

- ShortFormulaEncoder ()
- bool canEncode (const ConstraintT &constraint)
- Rational encodingSize (const ConstraintT &constraint)
- std::string name ()
- std::optional< FormulaT > encode (const ConstraintT &constraint)

  Encodes an arbitrary constraint.

Data Fields

- std::size_t problem_size

Protected Member Functions

- std::optional< FormulaT > doEncode (const ConstraintT &constraint)
- FormulaT generateVarChain (const std::set< carl::Variable > &vars, carl::FormulaType type)

0.15.470.1 Constructor & Destructor Documentation

0.15.470.1.1 ShortFormulaEncoder()

```
smtrat::ShortFormulaEncoder::ShortFormulaEncoder ( ) [inline]
```

0.15.470.2 Member Function Documentation

0.15.470.2.1 canEncode()

`bool smtrat::ShortFormulaEncoder::canEncode ( const ConstraintT & constraint ) [virtual]`

Implements smtrat::PseudoBoolEncoder.

0.15.470.2.2 doEncode()

`std::optional< FormulaT > smtrat::ShortFormulaEncoder::doEncode ( const ConstraintT & constraint ) [protected], [virtual]`

Implements smtrat::PseudoBoolEncoder.

0.15.470.2.3 encode()

`std::optional< FormulaT > smtrat::PseudoBoolEncoder::encode ( const ConstraintT & constraint ) [inherited]`

Encodes an arbitrary constraint.

Returns

encoded formula

0.15.470.2.4 encodingSize()

`Rational smtrat::ShortFormulaEncoder::encodingSize ( const ConstraintT & constraint ) [virtual]`

Reimplemented from smtrat::PseudoBoolEncoder.

0.15.470.2.5 generateVarChain()

`FormulaT smtrat::PseudoBoolEncoder::generateVarChain ( const std::set< carl::Variable > & vars, carl::FormulaType type ) [protected], [inherited]`

0.15.470.2.6 name()

`std::string smtrat::ShortFormulaEncoder::name ( ) [inline], [virtual]`

Reimplemented from smtrat::PseudoBoolEncoder.

0.15.470.3 Field Documentation

0.15.470.3.1 problem_size

`std::size_t smtrat::PseudoBoolEncoder::problem_size [inherited]`

0.15.471 benchmax::simple_parser Class Reference

`#include <SMTRAT.h>`

Public Member Functions

- simple_parser (const std::string &content)
- bool expect (const char c)
- bool expect_end ()
- void skip_whitespace ()
- void skip_excluding (const char c)
- std::optional< std::string > read_until (const char c)
- std::optional< std::string > read_until_whitespace ()
- std::optional< std::string > read_until_whitespace_or (const std::initializer_list< char > cs)

0.15.471.1 Constructor & Destructor Documentation

0.15.471.1.1 simple_parser()

```
benchmax::simple_parser::simple_parser (
    const std::string & content ) [inline]
```

0.15.471.2 Member Function Documentation

0.15.471.2.1 expect()

```
bool benchmax::simple_parser::expect (
    const char c ) [inline]
```

0.15.471.2.2 expect_end()

```
bool benchmax::simple_parser::expect_end ( ) [inline]
```

0.15.471.2.3 read_until()

```
std::optional<std::string> benchmax::simple_parser::read_until (
    const char c ) [inline]
```

0.15.471.2.4 read_until_whitespace()

```
std::optional<std::string> benchmax::simple_parser::read↩
_until_whitespace ( ) [inline]
```

0.15.471.2.5 read_until_whitespace_or()

```
std::optional<std::string> benchmax::simple_parser↩
::read_until_whitespace_or (
    const std::initializer_list< char > cs ) [inline]
```

0.15.471.2.6 skip_excluding()

```
void benchmax::simple_parser::skip_excluding (
    const char c ) [inline]
```

0.15.471.2.7 skip_whitespace()

```
void benchmax::simple_parser::skip_whitespace ( ) [inline]
```

0.15.472 smtrat::parser::Theories::SimpleSortAdder Struct Reference

Helper functor for addSimpleSorts() method.

`#include <Theories.h>`

Public Member Functions

- SimpleSortAdder (qi::symbols< char, carl::Sort > &sorts)
- template<typename T >
  void operator() (T *)

Data Fields

- qi::symbols< char, carl::Sort > & sorts

0.15.472.1 Detailed Description

Helper functor for addSimpleSorts() method.

0.15.472.2 Constructor & Destructor Documentation

0.15.472.2.1 SimpleSortAdder() `smtrat::parser::Theories::SimpleSortAdder::SimpleSortAdder ( qi::symbols< char, carl::Sort > & sorts ) [inline]`

0.15.472.3 Member Function Documentation

0.15.472.3.1 operator()() `template<typename T > void smtrat::parser::Theories::SimpleSortAdder::operator() ( T * ) [inline]`

0.15.472.4 Field Documentation

0.15.472.4.1 sorts `qi::symbols<char, carl::Sort>& smtrat::parser::Theories::SimpleSortAdder↩ ::sorts`

0.15.473 smtrat::parser::SimpleSymbolParser Struct Reference

Parses symbols: `simple_symbol` | `quoted_symbol` where.
`#include <Lexicon.h>`

Public Member Functions

- SimpleSymbolParser ()

Data Fields

- qi::rule< Iterator, std::string(), Skipper > main

0.15.473.1 Detailed Description

Parses symbols: `simple_symbol` | `quoted_symbol` where.

- `simple_symbol` is any string of `[0-9a-zA-Z~!@$%^&*_-+=<>.?/]` that does not start with a digit and is not a reserved word.
- `quoted_symbol` is any string of printable characters (including space, tab, line-breaks) except \ and | enclosed in | characters.

0.15.473.2 Constructor & Destructor Documentation

0.15.473.2.1 SimpleSymbolParser() `smtrat::parser::SimpleSymbolParser::SimpleSymbolParser ( ) [inline]`

0.15.473.3 Field Documentation

0.15.473.3.1 main `qi::rule<Iterator, std::string(), Skipper> smtrat::parser::SimpleSymbol↩ Parser::main`

0.15.474 smtrat::expression::simplifier::Simplifier Class Reference

`#include <Simplifier.h>`

Public Member Functions

- const ExpressionContent ∗ operator() (const ExpressionContent ∗_ec) const

0.15.474.1 Member Function Documentation

0.15.474.1.1 operator()()

```
const ExpressionContent* smtrat::expression::simplifier::Simplifier↩
::operator() (
            const ExpressionContent * _ec ) const  [inline]
```

0.15.475 smtrat::expression::simplifier::SimplifierChainCaller< chainID > Struct Template Reference

`#include <Simplifier.h>`

Public Member Functions

- const ExpressionContent ∗ operator() (const ExpressionContent ∗_ec, const SimplifierChain &_chain) const

Data Fields

- SimplifierChainCaller< chainID-1 > recurse

0.15.475.1 Member Function Documentation

0.15.475.1.1 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]
```

0.15.475.2 Field Documentation

0.15.475.2.1 recurse

```
template<std::size_t chainID = std::tuple_size<SimplifierChain>::value -
1>
SimplifierChainCaller<chainID-1> smtrat::expression::simplifier::SimplifierChainCaller< chain↩
ID >::recurse
```

0.15.476 smtrat::expression::simplifier::SimplifierChainCaller< 0 > Struct Reference

`#include <Simplifier.h>`

Public Member Functions

- const ExpressionContent ∗ operator() (const ExpressionContent ∗_ec, const SimplifierChain &_chain) const

0.15.476.1 Member Function Documentation

0.15.476.1.1 operator()()

```
const ExpressionContent* smtrat::expression::simplifier::SimplifierChainCaller<
0 >::operator() (
            const ExpressionContent * _ec,
            const SimplifierChain & _chain ) const  [inline]
```

0.15.477 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

0.15.477.1 Member Function Documentation

0.15.477.1.1 operator()() [1/7]

```
const ExpressionContent* smtrat::expression::simplifier::Base↩
Simplifier::operator() (
            const BinaryExpression & expr ) const  [inline], [inherited]
```

0.15.477.1.2 operator()() [2/7]

```
const ExpressionContent* smtrat::expression::simplifier::Base↩
Simplifier::operator() (
            const carl::Variable & expr ) const  [inline], [inherited]
```

0.15.477.1.3 operator()() [3/7]

```
const ExpressionContent* smtrat::expression::simplifier::Base↩
Simplifier::operator() (
            const ExpressionContent * _ec ) const  [inline], [inherited]
```

0.15.477.1.4 operator()() [4/7]

```
const ExpressionContent* smtrat::expression::simplifier::Base↩
Simplifier::operator() (
            const ITEExpression & expr ) const  [inline], [inherited]
```

0.15.477.1.5 operator()() [5/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const NaryExpression & expr ) const [inline], [inherited]
```

0.15.477.1.6 operator()() [6/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const QuantifierExpression & expr ) const [inline], [inherited]
```

0.15.477.1.7 operator()() [7/7]

```
const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::operator() (
            const UnaryExpression & expr ) const [inline], [inherited]
```

0.15.477.1.8 simplify() [1/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const BinaryExpression & ) const [inline], [protected], [virtual], [inherited]
```

0.15.477.1.9 simplify() [2/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const carl::Variable & ) const [inline], [protected], [virtual], [inherited]
```

0.15.477.1.10 simplify() [3/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const ITEExpression & ) const [inline], [protected], [virtual], [inherited]
```

0.15.477.1.11 simplify() [4/6]

```
const ExpressionContent* smtrat::expression::simplifier::SingletonSimplifier::simplify (
            const NaryExpression & expr ) const [inline], [virtual]
```

Reimplemented from smtrat::expression::simplifier::BaseSimplifier.

0.15.477.1.12 simplify() [5/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const QuantifierExpression & ) const [inline], [protected], [virtual], [inherited]
```

0.15.477.1.13 simplify() [6/6]

```
virtual const ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify (
            const UnaryExpression & ) const [inline], [protected], [virtual], [inherited]
```

Reimplemented in smtrat::expression::simplifier::NegationSimplifier.

0.15.478 smtrat::cad::sample_compare::size Struct Reference

```
#include <SampleComparator.h>
```

0.15.479 smtrat::parser::Skipper Struct Reference

```
#include <Common.h>
```

Public Member Functions

- Skipper ()

Data Fields

- boost::spirit::qi::rule< Iterator > main

0.15.479.1 Constructor & Destructor Documentation

0.15.479.1.1 Skipper() `smtrat::parser::Skipper::Skipper ( ) [inline]`

0.15.479.2 Field Documentation

0.15.479.2.1 main `boost::spirit::qi::rule<Iterator> smtrat::parser::Skipper::main`

0.15.480 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.

0.15.480.1 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.

0.15.480.2 Member Function Documentation

0.15.480.2.1 addResult()

```
void benchmax::Backend::addResult (
    const Tool * tool,
    const fs::path & file,
    BenchmarkResult && result ) [inline], [inherited]
```

Add a result.

0.15.480.2.2 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.

0.15.480.2.3 finalize()

```
virtual void benchmax::Backend::finalize ( ) [inline], [protected],
[virtual], [inherited]
```

Hook to allow for asynchronous backends to wait for jobs to terminate.

0.15.480.2.4 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.

0.15.480.2.5 process_results()

```
void benchmax::Backend::process_results (
    const Jobs & jobs,
    bool check_finished ) [inline], [inherited]
```

0.15.480.2.6 run()

```
void benchmax::SlurmBackend::run (
    const Jobs & jobs,
    bool wait_for_termination ) [inline]
```

Run all tools on all benchmarks using Slurm.

0.15.480.2.7 sanitize_results()

```
void benchmax::Backend::sanitize_results (
    const Jobs & jobs ) const [inline], [protected], [inherited]
```

0.15.480.2.8 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.

0.15.480.2.9 suspendable()

```
bool benchmax::SlurmBackend::suspendable ( ) const [inline]
```

0.15.480.2.10 write_results()

```
void benchmax::Backend::write_results (
    const Jobs & jobs ) const [inline], [protected], [inherited]
```

0.15.480.3 Field Documentation

0.15.480.3.1 mExpectedJobs `std::size_t benchmax::Backend::mExpectedJobs [protected], [inherited]`

Number of jobs that should be run.

0.15.480.3.2 mFinishedJobs `std::atomic<std::size_t> benchmax::Backend::mFinishedJobs [protected], [inherited]`

Number of jobs that are finished.

0.15.480.3.3 mLastPercent `std::atomic<std::size_t> benchmax::Backend::mLastPercent [protected], [inherited]`

Percentage of finished jobs when madeProgress() was last called.

0.15.481 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.

0.15.481.1 Detailed Description

Settings for the Slurm backend.

0.15.481.2 Field Documentation

0.15.481.2.1 archive_log_file `std::string benchmax::settings::SlurmBackendSettings::archive_log_file`

Puts logs to some archive.

0.15.481.2.2 array_size `std::size_t benchmax::settings::SlurmBackendSettings::array_size`

Number of array jobs within one job.

0.15.481.2.3 keep_logs `bool benchmax::settings::SlurmBackendSettings::keep_logs`

Do not remove logs from file system if set to true.

0.15.481.2.4 sbatch_options `std::string benchmax::settings::SlurmBackendSettings::sbatch_options`

Additional options passed on to slurm.

0.15.481.2.5 slice_size `std::size_t benchmax::settings::SlurmBackendSettings::slice_size`

Size of one slice that is handled in one array job.

0.15.481.2.6 submission_delay `carl::settings::duration benchmax::settings::SlurmBackendSettings::submission_delay`

Delay between job submissions.

0.15.481.2.7 tmp_dir `std::string benchmax::settings::SlurmBackendSettings::tmp_dir`

Temporary directory for output files.

0.15.482 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 ([[maybe_unused]] const std::vector< types::TermType > &vars)
- void addObjective (const types::TermType &t, OptimizationType ot)
- void pop (const Integer &n)
- void push (const Integer &n)
- void reset ()
- void resetAssertions ()
- void setInfo (const Attribute &attribute)
- void setLogic (const carl::Logic &name)
- void setOption (const Attribute &option)
- template<typename Function , typename... Args>
  void callHandler (const Function &f, const Args &... args)

Data Fields

- bool queueInstructions
- InstructionHandler & handler
- ParserState state
- Theories theories
- ScriptParser< SMTLIBParser > parser

0.15.482.1 Constructor & Destructor Documentation

0.15.482.1.1 SMTLIBParser()

```
smtrat::parser::SMTLIBParser::SMTLIBParser (
    InstructionHandler & handler,
    bool queueInstructions ) [inline]
```

0.15.482.1.2 ~SMTLIBParser()

```
smtrat::parser::SMTLIBParser::~SMTLIBParser ( ) [inline]
```

0.15.482.2 Member Function Documentation

0.15.482.2.1 add()

```
void smtrat::parser::SMTLIBParser::add (
    const types::TermType & t,
    bool isSoftFormula = false,
    Rational weight = Rational(1),
    const std::string id = std::string() ) [inline]
```

0.15.482.2.2 addObjective()

```
void smtrat::parser::SMTLIBParser::addObjective (
    const types::TermType & t,
    OptimizationType ot ) [inline]
```

0.15.482.2.3 callHandler()

```
template<typename Function , typename... Args>
void smtrat::parser::SMTLIBParser::callHandler (
    const Function & f,
    const Args &... args ) [inline]
```

0.15.482.2.4 check()

```
void smtrat::parser::SMTLIBParser::check ( ) [inline]
```

0.15.482.2.5 declareConst()

```
void smtrat::parser::SMTLIBParser::declareConst (
    const std::string & name,
    const carl::Sort & sort ) [inline]
```

0.15.482.2.6 declareFun()

```
void smtrat::parser::SMTLIBParser::declareFun (
    const std::string & name,
    const std::vector< carl::Sort > & args,
    const carl::Sort & sort ) [inline]
```

0.15.482.2.7 declareSort()

```
void smtrat::parser::SMTLIBParser::declareSort (
const std::string & name,
Integer arity ) [inline]
```

0.15.482.2.8 echo()

```
void smtrat::parser::SMTLIBParser::echo (
const std::string & s ) [inline]
```

0.15.482.2.9 eliminateQuantifiers()

```
void smtrat::parser::SMTLIBParser::eliminateQuantifiers (
const qe::QEQuery & q ) [inline]
```

0.15.482.2.10 exit()

```
void smtrat::parser::SMTLIBParser::exit ( ) [inline]
```

0.15.482.2.11 getAllModels()

```
void smtrat::parser::SMTLIBParser::getAllModels ( ) [inline]
```

0.15.482.2.12 getAssertions()

```
void smtrat::parser::SMTLIBParser::getAssertions ( ) [inline]
```

0.15.482.2.13 getAssignment()

```
void smtrat::parser::SMTLIBParser::getAssignment ( ) [inline]
```

0.15.482.2.14 getInfo()

```
void smtrat::parser::SMTLIBParser::getInfo (
const std::string & key ) [inline]
```

0.15.482.2.15 getModel()

```
void smtrat::parser::SMTLIBParser::getModel ( ) [inline]
```

0.15.482.2.16 getObjectives()

```
void smtrat::parser::SMTLIBParser::getObjectives ( ) [inline]
```

0.15.482.2.17 getOption()

```
void smtrat::parser::SMTLIBParser::getOption (
const std::string & key ) [inline]
```

0.15.482.2.18 getProof()

```
void smtrat::parser::SMTLIBParser::getProof ( ) [inline]
```

0.15.482.2.19 getUnsatCore()

```
void smtrat::parser::SMTLIBParser::getUnsatCore ( ) [inline]
```

0.15.482.2.20 getValue()

```
void smtrat::parser::SMTLIBParser::getValue (
[[maybe_unused] ] const std::vector< types::TermType > & vars ) [inline]
```

0.15.482.2.21 parse()

```
bool smtrat::parser::SMTLIBParser::parse (
std::istream & in ) [inline]
```

0.15.482.2.22 pop()

```
void smtrat::parser::SMTLIBParser::pop (
const Integer & n ) [inline]
```

0.15.482.2.23 push()

```
void smtrat::parser::SMTLIBParser::push (
const Integer & n ) [inline]
```

0.15.482.2.24 reset()

```
void smtrat::parser::SMTLIBParser::reset ( ) [inline]
```

0.15.482.2.25 resetAssertions()

```
void smtrat::parser::SMTLIBParser::resetAssertions ( ) [inline]
```

0.15.482.2.26 setInfo()

```
void smtrat::parser::SMTLIBParser::setInfo (
const Attribute & attribute ) [inline]
```

0.15.482.2.27 setLogic()

```
void smtrat::parser::SMTLIBParser::setLogic (
const carl::Logic & name ) [inline]
```

0.15.482.2.28 setOption()

```
void smtrat::parser::SMTLIBParser::setOption (
const Attribute & option ) [inline]
```

0.15.482.3 Field Documentation

0.15.482.3.1 handler

```
InstructionHandler& smtrat::parser::SMTLIBParser::handler
```

0.15.482.3.2 parser

```
ScriptParser<SMTLIBParser> smtrat::parser::SMTLIBParser::parser
```

0.15.482.3.3 queueInstructions

```
bool smtrat::parser::SMTLIBParser::queueInstructions
```

0.15.482.3.4 state

```
ParserState smtrat::parser::SMTLIBParser::state
```

0.15.482.3.5 theories

```
Theories smtrat::parser::SMTLIBParser::theories
```

0.15.483 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.

0.15.483.1 Detailed Description

Tool wrapper for SMT-RAT for SMT-LIB.

0.15.483.2 Constructor & Destructor Documentation

0.15.483.2.1 SMTRAT()

```
benchmax::SMTRAT::SMTRAT (
    const fs::path & binary,
    const std::string & arguments ) [inline]
```

New SMT-RAT tool.

0.15.483.3 Member Function Documentation

0.15.483.3.1 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.

0.15.483.3.2 attributeHash() `std::size_t benchmax::Tool::attributeHash ( ) const [inline], [inherited]`

Hash of the attributes.

0.15.483.3.3 attributes() `const std::map<std::string,std::string>& benchmax::Tool::attributes ( ) const [inline], [inherited]`

A set of attributes, for example compilation options.

0.15.483.3.4 binary() `fs::path benchmax::Tool::binary ( ) const [inline], [inherited]`

Full path to the binary.

0.15.483.3.5 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.

0.15.483.3.6 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.

0.15.483.3.7 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.

0.15.483.3.8 getStatusFromOutput() `std::string benchmax::SMTRAT::getStatusFromOutput ( const BenchmarkResult & result ) const [inline]`

Try to parse memouts from stderr.

0.15.483.3.9 name() `std::string benchmax::Tool::name ( ) const [inline], [inherited]`

Common name of this tool.

0.15.483.3.10 parse_stats() `bool benchmax::SMTRAT::parse_stats ( BenchmarkResult & result ) const [inline]`

0.15.483.3.11 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.

0.15.483.3.12 resolveDependencies() `std::vector<std::string> benchmax::Tool::resolveDependencies ( ) const [inline], [inherited]`

Get dependencies of binary required to run it (via ldd)

0.15.483.4 Field Documentation

0.15.483.4.1 mArguments `std::string benchmax::Tool::mArguments [protected], [inherited]`

Command line arguments that should be passed to the binary.

0.15.483.4.2 mAttributes `std::map<std::string,std::string> benchmax::Tool::mAttributes [protected], [inherited]`

Attributes of the tool obtained by introspection of the binary.

0.15.483.4.3 mBinary `fs::path benchmax::Tool::mBinary [protected], [inherited]`

Path to the binary.

0.15.483.4.4 mName `std::string benchmax::Tool::mName [protected], [inherited]`

(Non-unique) identifier for the tool.

0.15.484 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.

0.15.484.1 Detailed Description

Tool wrapper for SMT-RAT for SMT-LIB.

0.15.484.2 Constructor & Destructor Documentation

0.15.484.2.1 SMTRAT_Analyzer()

```
benchmax::SMTRAT_Analyzer::SMTRAT_Analyzer (
    const fs::path & binary,
    const std::string & arguments ) [inline]
```

New SMT-RAT tool.

0.15.484.3 Member Function Documentation

0.15.484.3.1 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.

0.15.484.3.2 attributeHash()

```
std::size_t benchmax::Tool::attributeHash ( ) const [inline], [inherited]
```

Hash of the attributes.

0.15.484.3.3 attributes()

```
const std::map<std::string,std::string>& benchmax::Tool::attributes ( ) const [inline], [inherited]
```

A set of attributes, for example compilation options.

0.15.484.3.4 binary()

```
fs::path benchmax::Tool::binary ( ) const [inline], [inherited]
```

Full path to the binary.

0.15.484.3.5 canHandle()

```
virtual bool benchmax::SMTRAT_Analyzer::canHandle (
    const fs::path & path ) const [inline], [override], [virtual]
```

Only handle .smt2 files.
Reimplemented from benchmax::Tool.

0.15.484.3.6 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.

0.15.484.3.7 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.

0.15.484.3.8 name()

`std::string benchmax::Tool::name ( ) const [inline], [inherited]`

Common name of this tool.

0.15.484.3.9 parseCommandline()

`virtual std::optional<std::string> benchmax::Tool::parseCommandline ( const std::string & cmdline ) const [inline], [virtual], [inherited]`

Compose commandline for this tool and the given input file.

0.15.484.3.10 resolveDependencies()

`std::vector<std::string> benchmax::Tool::resolveDependencies ( ) const [inline], [inherited]`

Get dependencies of binary required to run it (via ldd)

0.15.484.4 Field Documentation

0.15.484.4.1 mArguments

`std::string benchmax::Tool::mArguments [protected], [inherited]`

Command line arguments that should be passed to the binary.

0.15.484.4.2 mAttributes

`std::map<std::string,std::string> benchmax::Tool::mAttributes [protected], [inherited]`

Attributes of the tool obtained by introspection of the binary.

0.15.484.4.3 mBinary

`fs::path benchmax::Tool::mBinary [protected], [inherited]`

Path to the binary.

0.15.484.4.4 mName

`std::string benchmax::Tool::mName [protected], [inherited]`

(Non-unique) identifier for the tool.

0.15.485 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.

0.15.485.1 Detailed Description

Adapts the SMTRAT solver for OPB files.

0.15.485.2 Constructor & Destructor Documentation

0.15.485.2.1 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.

0.15.485.3 Member Function Documentation

0.15.485.3.1 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.

0.15.485.3.2 attributeHash() `std::size_t benchmax::Tool::attributeHash ( ) const [inline], [inherited]`

Hash of the attributes.

0.15.485.3.3 attributes() `const std::map<std::string,std::string>& benchmax::Tool::attributes ( ) const [inline], [inherited]`

A set of attributes, for example compilation options.

0.15.485.3.4 binary() `fs::path benchmax::Tool::binary ( ) const [inline], [inherited]`

Full path to the binary.

0.15.485.3.5 canHandle() `virtual bool benchmax::SMTRAT_OPB::canHandle ( const fs::path & path ) const [inline], [override], [virtual]`

Only handles .opb files.

Reimplemented from benchmax::SMTRAT.

0.15.485.3.6 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.

0.15.485.3.7 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.

0.15.485.3.8 getStatusFromOutput() `std::string benchmax::SMTRAT::getStatusFromOutput ( const BenchmarkResult & result ) const [inline], [inherited]`

Try to parse memouts from stderr.

0.15.485.3.9 name() `std::string benchmax::Tool::name ( ) const [inline], [inherited]`

Common name of this tool.

0.15.485.3.10 parse_stats() `bool benchmax::SMTRAT::parse_stats ( BenchmarkResult & result ) const [inline], [inherited]`

0.15.485.3.11 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.

0.15.485.3.12 resolveDependencies() `std::vector<std::string> benchmax::Tool::resolveDependencies ( ) const [inline], [inherited]`

Get dependencies of binary required to run it (via ldd)

0.15.485.4 Field Documentation

0.15.485.4.1 mArguments `std::string benchmax::Tool::mArguments [protected], [inherited]`

Command line arguments that should be passed to the binary.

0.15.485.4.2 mAttributes `std::map<std::string,std::string> benchmax::Tool::mAttributes [protected], [inherited]`

Attributes of the tool obtained by introspection of the binary.

0.15.485.4.3 mBinary `fs::path benchmax::Tool::mBinary [protected], [inherited]`

Path to the binary.

0.15.485.4.4 mName `std::string benchmax::Tool::mName [protected], [inherited]`

(Non-unique) identifier for the tool.

0.15.486 smtrat::execution::SoftAssertion Struct Reference

`#include <ExecutionState.h>`

Data Fields

- FormulaT formula
- Rational weight
- std::string id

0.15.486.1 Field Documentation

0.15.486.1.1 formula `FormulaT smtrat::execution::SoftAssertion::formula`

0.15.486.1.2 id `std::string smtrat::execution::SoftAssertion::id`

0.15.486.1.3 weight `Rational smtrat::execution::SoftAssertion::weight`

0.15.487 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

0.15.487.1 Field Documentation

0.15.487.1.1 convert_to_cnf_dimacs `bool smtrat::settings::SolverSettings::convert_to_cnf_dimacs`

0.15.487.1.2 convert_to_cnf_smtlib `bool smtrat::settings::SolverSettings::convert_to_cnf_smtlib`

0.15.487.1.3 preprocess `bool smtrat::settings::SolverSettings::preprocess`

0.15.487.1.4 preprocess_output_file `std::string smtrat::settings::SolverSettings::preprocess_output_file`

0.15.487.1.5 print_all_models `bool smtrat::settings::SolverSettings::print_all_models`

0.15.487.1.6 print_model `bool smtrat::settings::SolverSettings::print_model`

0.15.488 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

0.15.488.1 Constructor & Destructor Documentation

0.15.488.1.1 SortedVariableParser() `smtrat::parser::SortedVariableParser::SortedVariableParser ( ) [inline]`

0.15.488.2 Field Documentation

0.15.488.2.1 main `qi::rule<Iterator, std::pair<std::string, carl::Sort>), Skipper> smtrat::parser::SortedVariableParser::main`

0.15.488.2.2 sort `SortParser smtrat::parser::SortedVariableParser::sort`

0.15.488.2.3 symbol `SymbolParser smtrat::parser::SortedVariableParser::symbol`

0.15.489 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)

0.15.489.1 Constructor & Destructor Documentation

0.15.489.1.1 SortParser() `smtrat::parser::SortParser::SortParser ( ) [inline]`

0.15.489.2 Member Function Documentation

0.15.489.2.1 clearParameters() `void smtrat::parser::SortParser::clearParameters ( ) [inline]`

0.15.489.2.2 getSort() `carl::Sort smtrat::parser::SortParser::getSort ( const Identifier & i ) [inline]`

0.15.489.2.3 getSortWithParam() `carl::Sort smtrat::parser::SortParser::getSortWithParam ( const Identifier & i, const std::vector< carl::Sort > & params ) [inline]`

0.15.489.2.4 setParameters() `void smtrat::parser::SortParser::setParameters ( const std::vector< std::string > & params ) [inline]`

0.15.490 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

0.15.490.1 Constructor & Destructor Documentation

0.15.490.1.1 SpecConstantParser() `smtrat::parser::SpecConstantParser::SpecConstantParser ( ) [inline]`

0.15.490.2 Field Documentation

0.15.490.2.1 binary `BinaryParser smtrat::parser::SpecConstantParser::binary`

0.15.490.2.2 decimal `DecimalParser smtrat::parser::SpecConstantParser::decimal`

0.15.490.2.3 hexadecimal `HexadecimalParser smtrat::parser::SpecConstantParser::hexadecimal`

0.15.490.2.4 main `qi::rule<Iterator, types::ConstType(), Skipper> smtrat::parser::Spec↩ ConstantParser::main`

0.15.490.2.5 numeral `NumeralParser smtrat::parser::SpecConstantParser::numeral`

0.15.490.2.6 string `StringParser smtrat::parser::SpecConstantParser::string`

0.15.490.2.7 theoryConst `qi::symbols<char, types::ConstType> smtrat::parser::SpecConstant↩ Parser::theoryConst`

0.15.491 smtrat::SplitSOSModule< Settings > Class Template Reference

`#include <SplitSOSModule.h>`

Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- SplitSOSModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=NULL)
- ~SplitSOSModule ()
- Answer checkCore ()

  Checks the received formula for consistency.
- bool isPreprocessor () const
- bool appliedPreprocessing () const
- bool add (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- void remove (ModuleInput::const_iterator _subformula)

  Removes everything related to the given sub-formula of the received formula.
- Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)

  Checks the received formula for consistency.
- Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.

- virtual Answer runBackends ()
- std::pair< bool, FormulaT > getReceivedFormulaSimplified ()
- void updateModel () const

  Updates the current assignment into the model.
- bool inform (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void deinform (const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual void init ()

  Informs all backends about the so far encountered constraints, which have not yet been communicated.
- virtual void updateAllModels ()

  Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
- virtual std::list< std::vector< carl::Variable > > getModelEqualities () const

  Partition the variables from the current model into equivalence classes according to their assigned value.
- unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const
- virtual unsigned currentlySatisfied (const FormulaT &) const
- bool receivedVariable (carl::Variable::Arg _var) const
- Answer solverState () const
- std::size_t id () const
- void setId (std::size_t _id)

  Sets this modules unique ID to identify itself.
- thread_priority threadPriority () const
- void setThreadPriority (thread_priority _threadPriority)

  Sets the priority of this module to get a thread for running its check procedure.
- const ModuleInput * pReceivedFormula () const
- const ModuleInput & rReceivedFormula () const
- const ModuleInput * pPassedFormula () const
- const ModuleInput & rPassedFormula () const
- const Model & model () const
- const std::vector< Model > & allModels () const
- const std::vector< FormulaSetT > & infeasibleSubsets () const
- const std::vector< Module * > & usedBackends () const
- const carl::FastSet< FormulaT > & constraintsToInform () const
- const carl::FastSet< FormulaT > & informedConstraints () const
- void addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.
- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)
- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

  Copies the infeasible subsets of the passed formula.
- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()

- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations

- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.491.1 Member Typedef Documentation

0.15.491.1.1 SettingsType

```
template<typename Settings >
typedef Settings smtrat::SplitSOSModule< Settings >::SettingsType
```

0.15.491.2 Member Enumeration Documentation

0.15.491.2.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

0.15.491.3 Constructor & Destructor Documentation

0.15.491.3.1 SplitSOSModule()

```
template<typename Settings >
smtrat::SplitSOSModule< Settings >::SplitSOSModule (
            const ModuleInput * _formula,
```

```
Conditionals & _conditionals,
Manager * _manager = NULL )
```

0.15.491.3.2 ~SplitSOSModule() template<typename Settings >

smtrat::SplitSOSModule< Settings >::~SplitSOSModule ( )

0.15.491.4 Member Function Documentation

0.15.491.4.1 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.

0.15.491.4.2 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 >.

0.15.491.4.3 addCore() virtual bool smtrat::Module::addCore ([[maybe_unused]] 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.

0.15.491.4.4 addInformationRelevantFormula() void smtrat::Module::addInformationRelevantFormula (const FormulaT & formula) [protected], [inherited]

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.491.4.5 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.

0.15.491.4.6 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.

0.15.491.4.7 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.

0.15.491.4.8 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.

0.15.491.4.9 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.

0.15.491.4.10 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.

0.15.491.4.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const  [inline],
[inherited]
```

Returns

All satisfying assignments, if existent.

0.15.491.4.12 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.

0.15.491.4.13 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.

0.15.491.4.14 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.

0.15.491.4.15 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.

0.15.491.4.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.491.4.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.491.4.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
```

```
const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline],
[protected], [inherited]
```

0.15.491.4.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.491.4.20 check()

```
Answer smtrat::PModule::check (
    bool _final = false,
    bool _full = true,
    carl::Variable _objective = carl::Variable::NO_VARIABLE ) [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.491.4.21 checkCore()

```
template<typename 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.

0.15.491.4.22 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.

0.15.491.4.23 checkModel()

```
unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]
```

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.491.4.24 cleanModel()

```
void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]
```

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.491.4.25 clearLemmas()

```
void smtrat::Module::clearLemmas ( ) [inline], [inherited]
```

Deletes all yet found lemmas.

0.15.491.4.26 clearModel()

```
void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]
```

Clears the assignment, if any was found.

0.15.491.4.27 clearModels()

```
void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]
```

Clears all assignments, if any was found.

0.15.491.4.28 clearPassedFormula()

```
void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]
```

0.15.491.4.29 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

0.15.491.4.30 collectOrigins() [2/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.
If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.491.4.31 collectTheoryPropagations()

```
void smtrat::Module::collectTheoryPropagations ( ) [inherited]
```

0.15.491.4.32 constraintsToInform()

```
const carl::FastSet<FormulaT>& smtrat::Module::constraints↩
ToInform ( ) const [inline], [inherited]
```

Returns

The constraints which the backends must be informed about.

0.15.491.4.33 currentlySatisfied()

```
virtual unsigned smtrat::Module::currentlySatisfied (
    const FormulaT & ) const [inline], [virtual], [inherited]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.491.4.34 currentlySatisfiedByBackend()

```
unsigned smtrat::Module::currentlySatisfiedByBackend (
    const FormulaT & _formula ) const [inherited]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.491.4.35 deinform()

```
void smtrat::Module::deinform (
    const FormulaT & _constraint ) [inherited]
```

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.491.4.36 deinformCore()

`virtual void smtrat::Module::deinformCore ( [[maybe_unused] ] 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.

0.15.491.4.37 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

0.15.491.4.38 eraseSubformulaFromPassedFormula()

`ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula ( ModuleInput::iterator _subformula, bool _ignoreOrigins = false ) [protected], [virtual], [inherited]`

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_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 >.

0.15.491.4.39 excludeNotReceivedVariablesFromModel()

`void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

0.15.491.4.40 findBestOrigin()

`std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin ( const std::vector< FormulaT > & _origins ) const [protected], [inherited]`

Parameters

_origins	

Returns

0.15.491.4.41 firstSubformulaToPass()

`ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.491.4.42 firstUncheckedReceivedSubformula()

`ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.491.4.43 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.491.4.44 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.491.4.45 getBackendsAllModels()

`void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.491.4.46 getBackendsModel()

`void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.491.4.47 getInfeasibleSubsets() [1/2]

`void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.491.4.48 getInfeasibleSubsets() [2/2]

```
std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets (
            const Module & _backend ) const  [protected], [inherited]
```

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

Parameters

_backend	The backend from which to obtain the infeasible subsets.

Returns

The infeasible subsets the given backend provides.

0.15.491.4.49 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.491.4.50 getModelEqualities()

`std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.491.4.51 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
const FormulaT & _formula,
FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.491.4.52 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
const FormulaT & _formula,
FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.491.4.53 getOrigins() [3/3]

```
const FormulaT& smtrat::Module::getOrigins (
ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]
```

Gets the origins of the passed formula at the given position.

Parameters

_formula	The position of a formula in the passed formulas.

Returns

The origins of the passed formula at the given position.

0.15.491.4.54 getReceivedFormulaSimplified() `std::pair< bool, FormulaT > smtrat::PModule::getReceivedFormulaSimplified ( ) [virtual], [inherited]`

Returns

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's reveived formula, if the Boolean is true.

Reimplemented from smtrat::Module.

0.15.491.4.55 hasLemmas() `bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

0.15.491.4.56 hasValidInfeasibleSubset() `bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]`

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.491.4.57 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.491.4.58 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.491.4.59 inform() `bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.
It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

Parameters

_constraint	The constraint to inform about.

Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

0.15.491.4.60 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.

0.15.491.4.61 informCore()

`virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.491.4.62 informedConstraints()

`const carl::FastSet<FormulaT>& smtrat::Module::informedConstraints ( ) const [inline], [inherited]`

Returns

The position of the first constraint of which no backend has been informed about.

0.15.491.4.63 init()

`void smtrat::Module::init ( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not be called twice and only before the first runBackends call.
Reimplemented in smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntBlastModule< Settings >, smtrat::FPPModule< Settings >, smtrat::CurryModule< Settings >, and smtrat::BVModule< Settings >.

0.15.491.4.64 is_minimizing()

`bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.491.4.65 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.

0.15.491.4.66 isPreprocessor() `bool smtrat::PModule::isPreprocessor ( ) const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.491.4.67 lemmas() `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.491.4.68 merge() `std::vector< FormulaT > smtrat::Module::merge ( const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.
({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

Parameters

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns

The vector being the two given vectors merged.

0.15.491.4.69 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.491.4.70 modelsDisjoint() `bool smtrat::Module::modelsDisjoint ( const Model & _modelA, const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

Parameters

_modelA	The first model to check for.
_modelB	The second model to check for.

Returns

true, if there is no variable assigned by both models; false, otherwise.

0.15.491.4.71 moduleName()

`template<typename Settings > std::string smtrat::SplitSOSModule< Settings >::moduleName ( ) const [inline], [virtual]`

Returns

The name of the given module type as name.

Reimplemented from smtrat::Module.

0.15.491.4.72 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

0.15.491.4.73 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.491.4.74 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.491.4.75 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.491.4.76 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.491.4.77 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.491.4.78 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.491.4.79 printAllModels()

`void smtrat::Module::printAllModels ( std::ostream & _out = std::cout ) [inherited]`

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.491.4.80 printInfeasibleSubsets()

`void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.491.4.81 printModel()

`void smtrat::Module::printModel ( std::ostream & _out = std::cout ) const [inherited]`

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.491.4.82 printPassedFormula()

`void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.491.4.83 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.491.4.84 probablyLooping()

```
bool smtrat::Module::probablyLooping (
    const typename Poly::PolyType & _branchingPolynomial,
    const Rational & _branchingValue ) const [protected], [inherited]
```

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.491.4.85 receivedFormulaChecked()

```
void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]
```

Notifies that the received formulas has been checked.

0.15.491.4.86 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩InfeasibleSubset (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

0.15.491.4.87 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const [inline], [inherited]
```

0.15.491.4.88 remove()

```
void smtrat::PModule::remove (
    ModuleInput::const_iterator _subformula ) [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.
However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

Parameters

_subformula	The sub formula of the received formula to remove.

Reimplemented from smtrat::Module.

0.15.491.4.89 removeCore()

```
virtual void smtrat::Module::removeCore (
    [[maybe_unused] ] 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.

0.15.491.4.90 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin (
            ModuleInput::iterator _formula,
            const FormulaT & _origin ) [inline], [protected], [inherited]
```

0.15.491.4.91 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]
```

0.15.491.4.92 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.491.4.93 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]
```

Returns

A reference to the formula passed to this module.

0.15.491.4.94 runBackends() [1/2]

```
virtual Answer smtrat::PModule::runBackends ( ) [inline], [virtual], [inherited]
```

Reimplemented from smtrat::Module.

0.15.491.4.95 runBackends() [2/2]

```
Answer smtrat::PModule::runBackends (
            bool _final,
            bool _full,
            carl::Variable _objective ) [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

Parameters

_final	true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT.
_full	false, if this module should avoid too expensive procedures and rather return unknown instead.
_objective	if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good.

Returns

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.491.4.96 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.

0.15.491.4.97 setThreadPriority()

```
void smtrat::Module::setThreadPriority ( thread_priority _threadPriority ) [inline], [inherited]
```

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.491.4.98 solverState()

```
Answer smtrat::Module::solverState ( ) const [inline], [inherited]
```

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.491.4.99 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 !=.

0.15.491.4.100 threadPriority()

```
thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]
```

Returns

The priority of this module to get a thread for running its check procedure.

0.15.491.4.101 updateAllModels()

```
void smtrat::Module::updateAllModels ( ) [virtual], [inherited]
```

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.491.4.102 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.491.4.103 updateModel() `void smtrat::PModule::updateModel ( ) const [virtual], [inherited]`

Updates the current assignment into the model.
Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.
Reimplemented from smtrat::Module.

0.15.491.4.104 usedBackends() `const std::vector<Module*>& smtrat::Module::usedBackends ( ) const [inline], [inherited]`

Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.491.5 Field Documentation

0.15.491.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.491.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.491.5.3 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.

0.15.491.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.491.5.5 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

0.15.491.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.491.5.7 mFirstSubformulaToPass `ModuleInput::iterator smtrat::Module::mFirstSubformulaTo↩Pass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.491.5.8 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.

0.15.491.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.491.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.491.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.491.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.491.5.13 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.

0.15.491.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.491.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.491.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.491.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.491.5.18 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.

0.15.491.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]`

Reusable splitting variables.

0.15.491.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.491.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.491.5.22 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.

0.15.491.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

0.15.491.5.24 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).

0.15.491.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.492 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.

0.15.492.1 Constructor & Destructor Documentation

0.15.492.1.1 SSHBackend() `benchmax::SSHBackend::SSHBackend ( ) [inline]`

0.15.492.1.2 ~SSHBackend() `benchmax::SSHBackend::~SSHBackend ( ) [inline]`

0.15.492.2 Member Function Documentation

0.15.492.2.1 addResult()

```
void benchmax::Backend::addResult (
    const Tool * tool,
    const fs::path & file,
    BenchmarkResult && result ) [inline], [inherited]
```

Add a result.

0.15.492.2.2 collect_results()

```
virtual bool benchmax::Backend::collect_results (
    const Jobs & ,
    bool ) [inline], [protected], [virtual], [inherited]
```

0.15.492.2.3 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.

0.15.492.2.4 finalize()

```
virtual void benchmax::Backend::finalize ( ) [inline], [protected],
[virtual], [inherited]
```

Hook to allow for asynchronous backends to wait for jobs to terminate.

0.15.492.2.5 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.

0.15.492.2.6 process_results()

```
void benchmax::Backend::process_results (
    const Jobs & jobs,
    bool check_finished ) [inline], [inherited]
```

0.15.492.2.7 run()

```
void benchmax::SSHBackend::run (
    const Jobs & ,
    bool ) [inline]
```

Dummy if SSH is disabled.

0.15.492.2.8 sanitize_results()

```
void benchmax::Backend::sanitize_results (
    const Jobs & jobs ) const [inline], [protected], [inherited]
```

0.15.492.2.9 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.

0.15.492.2.10 suspendable()

```
bool benchmax::Backend::suspendable ( ) const [inline], [inherited]
```

0.15.492.2.11 write_results()

```
void benchmax::Backend::write_results (
    const Jobs & jobs ) const [inline], [protected], [inherited]
```

0.15.492.3 Field Documentation

0.15.492.3.1 mExpectedJobs

```
std::size_t benchmax::Backend::mExpectedJobs [protected], [inherited]
```

Number of jobs that should be run.

0.15.492.3.2 mFinishedJobs

```
std::atomic<std::size_t> benchmax::Backend::mFinishedJobs [protected],
[inherited]
```

Number of jobs that are finished.

0.15.492.3.3 mLastPercent

```
std::atomic<std::size_t> benchmax::Backend::mLastPercent [protected],
[inherited]
```

Percentage of finished jobs when madeProgress() was last called.

0.15.493 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.

0.15.493.1 Detailed Description

Settings for SSH backend.

0.15.493.2 Field Documentation

0.15.493.2.1 basedir `std::string benchmax::settings::SSHBackendSettings::basedir`

Base directory for solvers.

0.15.493.2.2 nodes `std::vector<std::string> benchmax::settings::SSHBackendSettings::nodes`

List of nodes to connect to.

0.15.493.2.3 resolve_deps `bool benchmax::settings::SSHBackendSettings::resolve_deps`

Resolve and upload dependencies of binary.

0.15.493.2.4 tmpdir `std::string benchmax::settings::SSHBackendSettings::tmpdir`

Temporary directory for benchmarks and output files.

0.15.493.2.5 use_wallclock `bool benchmax::settings::SSHBackendSettings::use_wallclock`

Use wallclock timeouts instead of CPU time.

0.15.494 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_IRUSR|S_IWUSR)

  Upload a file to the remote.
- bool executeCommand (const std::string &cmd, BenchmarkResult &result)

  Execute a command on the remote.

0.15.494.1 Detailed Description

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

0.15.494.2 Constructor & Destructor Documentation

0.15.494.2.1 SSHConnection() `benchmax::ssh::SSHConnection::SSHConnection ( const Node & n ) [inline]`

Create a new connection for the given node.

0.15.494.2.2 ~SSHConnection() `benchmax::ssh::SSHConnection::~SSHConnection ( ) [inline]`

Wait for all channels to terminate.

0.15.494.3 Member Function Documentation

0.15.494.3.1 busy() `bool benchmax::ssh::SSHConnection::busy ( ) [inline]`

Check if all channels are busy.

0.15.494.3.2 createTmpDir() `std::string benchmax::ssh::SSHConnection::createTmpDir ( const std::string & folder ) [inline]`

Create a temporary directory on the remote.

0.15.494.3.3 executeCommand() `bool benchmax::ssh::SSHConnection::executeCommand ( const std::string & cmd, BenchmarkResult & result ) [inline]`

Execute a command on the remote.

0.15.494.3.4 finishJob() `void benchmax::ssh::SSHConnection::finishJob ( ) [inline]`

Decrease job counter.

0.15.494.3.5 getNode() `const Node& benchmax::ssh::SSHConnection::getNode ( ) const [inline]`

Return the node.

0.15.494.3.6 jobFree() `bool benchmax::ssh::SSHConnection::jobFree ( ) [inline]`

Check if a new job could be started.

0.15.494.3.7 jobs() `std::size_t benchmax::ssh::SSHConnection::jobs ( ) const [inline]`

Current number of jobs.

0.15.494.3.8 newJob() `void benchmax::ssh::SSHConnection::newJob ( ) [inline]`

Increase job counter.

0.15.494.3.9 removeDir() `void benchmax::ssh::SSHConnection::removeDir ( const std::string & folder ) [inline]`

Remove a (temporary) directory on the remote.

0.15.494.3.10 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.

0.15.495 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 _withVariableBounds)

  Constructs a state being a child of the given state and containing the given substitution, which maps from the index variable of the given state.
- State (const State &)=delete
- ~State ()

  Destructor.
- bool isRoot () const
- bool cannotBeSolved (bool _lazy) const
- bool & rCannotBeSolved ()
- bool & rCannotBeSolvedLazy ()
- bool markedAsDeleted () const
- bool & rMarkedAsDeleted ()
- bool hasChildrenToInsert () const
- bool & rHasChildrenToInsert ()
- carl::Variable::Arg index () const
- size_t valuation () const
- size_t backendCallValuation () const
- size_t id () const
- size_t & rID ()

- std::list< State * > & rChildren ()
- const std::list< State * > & children () const
- State * pFather () const
- const State & father () const
- State & rFather ()
- ConflictSets & rConflictSets ()
- const ConflictSets & conflictSets () const
- bool & rHasRecentlyAddedConditions ()
- bool hasRecentlyAddedConditions () const
- bool & rInconsistent ()
- bool isInconsistent () const
- ConditionList & rConditions ()
- const ConditionList & conditions () const
- Substitution & rSubstitution ()
- const Substitution & substitution () const
- SubstitutionResults & rSubstitutionResults ()
- const SubstitutionResults & substitutionResults () const
- SubResultCombination & rSubResultCombination ()
- const SubResultCombination & subResultCombination () const
- const Substitution * pSubstitution () const
- bool conditionsSimplified () const
- bool subResultsSimplified () const
- bool & rSubResultsSimplified ()
- bool takeSubResultCombAgain () const
- bool & rTakeSubResultCombAgain ()
- bool tryToRefreshIndex () const
- bool hasSubResultsCombination () const
- bool hasSubstitutionResults () const
- bool unfinished () const
- Type type () const
- Type & rType ()
- const Condition * pOriginalCondition () const
- const Condition & originalCondition () const
- const carl::PointerSet< Condition > & tooHighDegreeConditions () const
- carl::PointerSet< Condition > & rTooHighDegreeConditions ()
- const VariableBoundsCond & variableBounds () const
- VariableBoundsCond & rVariableBounds ()
- void setOriginalCondition (const Condition *_pOCondition)

  Sets the pointer of the original condition of this state to the given pointer to a condition.
- size_t currentRangeSize () const
- void resetCurrentRangeSize ()
- const 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 * > &_↩reactivatedStates)

  Updates the original conditions of substitutions having the same test candidate as the given.
- void addSubstitutionResults (std::vector< DisjunctionOfConditionConjunctions > &&_disjunctionsOfCond↩Conj)

  Adds the given substitution results to this state.
- bool extendSubResultCombination ()

  Extends the currently considered combination of conjunctions in the substitution results.
- bool nextSubResultCombination ()

  If the state contains a substitution result, which is a conjunction of disjunctions of conjunctions of conditions, this method sets the current combination to the disjunctions to the next combination.
- size_t getNumberOfCurrentSubresultCombination () const
- ConditionList getCurrentSubresultCombination () const

  Gets the current substitution result combination as condition vector.
- bool refreshConditions (ValuationMap &_ranking)

Determines the condition vector corresponding to the current combination of the conjunctions of conditions.

- void initConditionFlags ()

  Sets all flags of the conditions to true, if it contains the variable given by the states index.
- bool initIndex (const carl::Variables &_allVariables, const 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 > &_originalConditions, size_t _valutation, bool _recently Added, ValuationMap &_ranking)

  Adds a constraint to the conditions of this state.
- bool checkSubresultCombinations () const

  This is just for debug purpose.
- int deleteOrigins (carl::PointerSet< Condition > &_originsToDelete, ValuationMap &_ranking)

  Removes everything in this state originated by the given vector of conditions.
- void deleteOriginsFromChildren (carl::PointerSet< Condition > &_originsToDelete, ValuationMap &_ranking)

  Deletes everything originated by the given conditions in the children of this state.
- void deleteOriginsFromConflictSets (carl::PointerSet< Condition > &_originsToDelete, bool _originsAreCurrentConditions)

  Deletes everything originated by the given conditions in the conflict sets of this state.
- void deleteOriginsFromSubstitutionResults (carl::PointerSet< Condition > &_originsToDelete)

  Deletes everything originated by the given conditions in the substitution results of this state.
- void deleteConditions (carl::PointerSet< Condition > &_conditionsToDelete, ValuationMap &_ranking)

  Delete everything originated by the given conditions from the entire subtree with this state as root.
- std::vector< State * > addChild (const Substitution &_substitution)

  Adds a state as child to this state with the given substitution.
- void updateValuation (bool _lazy)

  Updates the valuation of this state.
- void updateBackendCallValuation ()

  Valuates the state's currently considered conditions according to a backend call.
- void passConflictToFather (bool _checkConflictForSideCondition, bool _useBackjumping=true, bool _includeInconsistentTestCandidates=false)

  Passes the original conditions of the covering set of the conflicts of this state to its father.
- bool hasLocalConflict ()

  Checks whether the currently considered conditions, which have been considered for test candidate construction, form already a conflict.
- bool checkTestCandidatesForBounds ()

  Checks whether the test candidate of this state is valid against the variable intervals in the father of this state.
- std::vector< 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 > &_min↩ CovSet, unsigned _currentTreeDepth)

  Finds a covering set of a vector of sets of sets due to some heuristics.

0.15.495.1 Member Typedef Documentation

0.15.495.1.1 ConflictSets `typedef carl::FastPointerMapB<Substitution,ConditionSetSetSet> smtrat::vs::State::Co`

0.15.495.1.2 SubResultCombination `typedef std::vector<std::pair< unsigned, unsigned> > smtrat::vs::State::Su`

0.15.495.1.3 SubstitutionResult `typedef std::vector<std::pair< ConditionList, bool> > smtrat::vs::State::Subst`

0.15.495.1.4 SubstitutionResults `typedef std::vector<SubstitutionResult> smtrat::vs::State::SubstitutionResult`

0.15.495.1.5 VariableBoundsCond `typedef smtrat::vb::VariableBounds<const Condition*> smtrat::vs::State::Vari`

0.15.495.2 Member Enumeration Documentation

0.15.495.2.1 Type `enum smtrat::vs::State::Type`

Enumerator

TEST_CANDIDATE_TO_GENERATE	
SUBSTITUTION_TO_APPLY	
COMBINE_SUBRESULTS	

0.15.495.3 Constructor & Destructor Documentation

0.15.495.3.1 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.

0.15.495.3.2 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.

0.15.495.3.3 State() [3/3]

```
smtrat::vs::State::State (
const State & ) [delete]
```

0.15.495.3.4 ~State()

```
smtrat::vs::State::~State ( )
```

Destructor.

0.15.495.4 Member Function Documentation

0.15.495.4.1 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.

0.15.495.4.2 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.
_recently Added	Is the condition a recently added one. @sideeffect The state can obtain a new condition.

0.15.495.4.3 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.

0.15.495.4.4 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.

0.15.495.4.5 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.

0.15.495.4.6 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.

0.15.495.4.7 averageConstraintValuation()

```
void smtrat::vs::State::averageConstraintValuation (
    const std::vector< std::pair< carl::Variable, std::multiset< double >>> & _varVals )
```

0.15.495.4.8 backendCallValuation()

```
size_t smtrat::vs::State::backendCallValuation ( ) const [inline]
```

Returns

The valuation of the state's considered conditions for a possible backend call.

0.15.495.4.9 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.

0.15.495.4.10 bestConstraintValuation()

```
void smtrat::vs::State::bestConstraintValuation (
    const std::vector< std::pair< carl::Variable, std::multiset< double >>> & _varVals )
```

0.15.495.4.11 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.

0.15.495.4.12 checkSubresultCombinations()

```
bool smtrat::vs::State::checkSubresultCombinations ( ) const
```

This is just for debug purpose.

Returns

true, if no subresult combination is illegal.

0.15.495.4.13 checkTestCandidatesForBounds() `bool smtrat::vs::State::checkTestCandidatesFor↩Bounds ( )`

Checks whether the test candidate of this state is valid against the variable intervals in the father of this state.

Returns

true, if the test candidate of this state is valid against the variable intervals; false, otherwise.

0.15.495.4.14 children() `const std::list<State*>& smtrat::vs::State::children ( ) const [inline]`

Returns

A constant reference to the vector of the children of this state.

0.15.495.4.15 conditions() `const ConditionList& smtrat::vs::State::conditions ( ) const [inline]`

Returns

A constant reference to the currently considered conditions of this state.

0.15.495.4.16 conditionsSimplified() `bool smtrat::vs::State::conditionsSimplified ( ) const [inline]`

Returns

true, if the considered conditions of this state are already simplified.

0.15.495.4.17 conflictSets() `const ConflictSets& smtrat::vs::State::conflictSets ( ) const [inline]`

Returns

A constant reference to the conflict sets of this state.

0.15.495.4.18 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.

0.15.495.4.19 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.

0.15.495.4.20 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.

0.15.495.4.21 currentRangeSize()

```
size_t smtrat::vs::State::currentRangeSize ( ) const [inline]
```

0.15.495.4.22 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.

0.15.495.4.23 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.

0.15.495.4.24 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.

0.15.495.4.25 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.

0.15.495.4.26 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.

0.15.495.4.27 extendSubResultCombination()

```
bool smtrat::vs::State::extendSubResultCombination
( )
```

Extends the currently considered combination of conjunctions in the substitution results.

0.15.495.4.28 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.

0.15.495.4.29 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.

0.15.495.4.30 getNextIntTestCandidate()

```
bool smtrat::vs::State::getNextIntTestCandidate ( smtrat::Rational & _nextIntTestCandidate, size_t _maxIntRange )
```

0.15.495.4.31 getNumberOfCurrentSubresultCombination()

```
size_t smtrat::vs::State::getNumberOfCurrentSubresultCombination ( ) const
```

Returns

The number of the current substitution result combination.

0.15.495.4.32 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.

0.15.495.4.33 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.

0.15.495.4.34 hasFurtherUncheckedTestCandidates()

```
bool smtrat::vs::State::hasFurtherUncheckedTestCandidates ( ) const
```

Checks whether there exist more than one test candidate, which has still not been checked.

Returns

true, if there exist more than one test candidate, which has still not been checked; false, otherwise.

0.15.495.4.35 hasInfinityChild()

```
bool smtrat::vs::State::hasInfinityChild ( ) const [inline]
```

0.15.495.4.36 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.

0.15.495.4.37 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.

0.15.495.4.38 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.

0.15.495.4.39 hasRecentlyAddedConditions() `bool smtrat::vs::State::hasRecentlyAddedConditions ( ) const [inline]`

Returns

true, if this state has a recently added condition in its considered conditions.

0.15.495.4.40 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.

0.15.495.4.41 hasSubResultsCombination() `bool smtrat::vs::State::hasSubResultsCombination ( ) const [inline]`

Returns

true, if this state has a substitution result combination.

0.15.495.4.42 hasSubstitutionResults() `bool smtrat::vs::State::hasSubstitutionResults ( ) const [inline]`

Returns

true, if this state has substitution results.

0.15.495.4.43 id() `size_t smtrat::vs::State::id ( ) const [inline]`

Returns

The id of this state.

0.15.495.4.44 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.

0.15.495.4.45 initConditionFlags() `void smtrat::vs::State::initConditionFlags ( )`

Sets all flags of the conditions to true, if it contains the variable given by the states index.

0.15.495.4.46 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	

0.15.495.4.47 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.

0.15.495.4.48 isRoot() `bool smtrat::vs::State::isRoot ( ) const [inline]`

Returns

The root of the state tree where this state is part from.

0.15.495.4.49 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.

0.15.495.4.50 maxIntTestCandidate() `const smtrat::Rational& smtrat::vs::State::maxIntTest↩ Candidate ( ) const [inline]`

0.15.495.4.51 minIntTestCandidate() `const smtrat::Rational& smtrat::vs::State::minIntTest↩ Candidate ( ) const [inline]`

0.15.495.4.52 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.

0.15.495.4.53 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.

0.15.495.4.54 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.

0.15.495.4.55 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.

0.15.495.4.56 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.

0.15.495.4.57 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.

0.15.495.4.58 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.

0.15.495.4.59 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.

0.15.495.4.60 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.

0.15.495.4.61 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.

0.15.495.4.62 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.

0.15.495.4.63 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.

0.15.495.4.64 printSubstitutionResultCombinationAsNumbers()

```
void smtrat::vs::State::print↩
SubstitutionResultCombinationAsNumbers (
            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.

0.15.495.4.65 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.

0.15.495.4.66 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.

0.15.495.4.67 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.

0.15.495.4.68 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.

0.15.495.4.69 rChildren() `std::list<State*>& smtrat::vs::State::rChildren ( ) [inline]`

Returns

A reference to the vector of the children of this state.

0.15.495.4.70 rConditions() `ConditionList& smtrat::vs::State::rConditions ( ) [inline]`

Returns

A reference to the currently considered conditions of this state.

0.15.495.4.71 rConflictSets() `ConflictSets& smtrat::vs::State::rConflictSets ( ) [inline]`

Returns

A reference to the conflict sets of this state.

0.15.495.4.72 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.

0.15.495.4.73 removeStatesFromRanking() `void smtrat::vs::State::removeStatesFromRanking ( const State & toRemove, ValuationMap & _ranking ) [static]`

0.15.495.4.74 resetCannotBeSolvedLazyFlags() `void smtrat::vs::State::resetCannotBeSolvedLazy↩ Flags ( )`

0.15.495.4.75 resetConflictSets() `void smtrat::vs::State::resetConflictSets ( )`

Clears the conflict sets.

0.15.495.4.76 resetCurrentRangeSize() `void smtrat::vs::State::resetCurrentRangeSize ( ) [inline]`

0.15.495.4.77 resetInfinityChild() `void smtrat::vs::State::resetInfinityChild ( const State * _state ) [inline]`

0.15.495.4.78 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.

0.15.495.4.79 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.

0.15.495.4.80 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.

0.15.495.4.81 rID() `size_t& smtrat::vs::State::rID ( ) [inline]`

Returns

A reference to the id of this state.

0.15.495.4.82 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.

0.15.495.4.83 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.

0.15.495.4.84 root() `State & smtrat::vs::State::root ( )`

Returns

The root of the tree, in which this state is located.

0.15.495.4.85 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())

0.15.495.4.86 rSubResultsSimplified() `bool& smtrat::vs::State::rSubResultsSimplified ( ) [inline]`

Returns

A reference to the flag indicating whether the substitution results of this state are already simplified.

0.15.495.4.87 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.

0.15.495.4.88 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())

0.15.495.4.89 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.

0.15.495.4.90 rTooHighDegreeConditions() `carl::PointerSet<Condition>& smtrat::vs::State::r↩ TooHighDegreeConditions ( ) [inline]`

Returns

A reference to the conditions of those this state currently considers, which have a too high degree to be tackled of the virtual substitution method.

0.15.495.4.91 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.

0.15.495.4.92 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.

0.15.495.4.93 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.

0.15.495.4.94 setInfinityChild() `void smtrat::vs::State::setInfinityChild ( State * _state ) [inline]`

0.15.495.4.95 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.

0.15.495.4.96 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.

0.15.495.4.97 simplify() [2/2] `void smtrat::vs::State::simplify ( ValuationMap & _ranking )`

Cleans up all conditions in this state according to comparison between the corresponding constraints.

0.15.495.4.98 solutionSpace()

```
std::vector< smtrat::DoubleInterval > smtrat::vs::State::solutionSpace (
    carl::PointerSet< Condition > & _conflictReason ) const
```

Determines the solution space of the test candidate of this state regarding to the variable bounds of the father. The solution space consists of one or two disjoint intervals.

Parameters

_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.

0.15.495.4.99 subResultCombination()

```
const SubResultCombination& smtrat::vs::State::subResultCombination ( ) const [inline]
```

Returns

A constant reference to the current combination of conjunction of constraints within the substitution results of this state. Make sure that a substitution result combination exists when calling this method. (using, e.g., hasSubResultsCombination())

0.15.495.4.100 subResultsSimplified()

```
bool smtrat::vs::State::subResultsSimplified ( ) const [inline]
```

Returns

true, if the substitution results of this state are already simplified.

0.15.495.4.101 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.

0.15.495.4.102 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.

0.15.495.4.103 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.

0.15.495.4.104 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())

0.15.495.4.105 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.

0.15.495.4.106 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.

0.15.495.4.107 treeDepth() `unsigned smtrat::vs::State::treeDepth ( ) const`

Returns

The depth of the subtree with this state as root node.

0.15.495.4.108 tryToRefreshIndex() `bool smtrat::vs::State::tryToRefreshIndex ( ) const [inline]`

Returns

true, if the index variable of this state shall be recalculated.

0.15.495.4.109 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.

0.15.495.4.110 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.

0.15.495.4.111 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.

0.15.495.4.112 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.

0.15.495.4.113 updateIntTestCandidates() `bool smtrat::vs::State::updateIntTestCandidates ( )`

0.15.495.4.114 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.

0.15.495.4.115 updateValuation() `void smtrat::vs::State::updateValuation ( bool _lazy )`

Updates the valuation of this state.

0.15.495.4.116 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).

0.15.495.4.117 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.

0.15.495.4.118 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.

0.15.495.4.119 worstConstraintValuation() `void smtrat::vs::State::worstConstraintValuation ( const std::vector< std::pair< carl::Variable, std::multiset< double >>> & _varVals )`

0.15.496 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

0.15.496.1 Member Typedef Documentation

0.15.496.1.1 Parents `using smtrat::StaticUnionFind::Parents = std::vector<Value>`

0.15.496.1.2 Ranks `using smtrat::StaticUnionFind::Ranks = std::vector<size_t>`

0.15.496.1.3 Representative `using smtrat::StaticUnionFind::Representative = Value`

0.15.496.1.4 Value `using smtrat::StaticUnionFind::Value = size_t`

0.15.496.2 Member Function Documentation

0.15.496.2.1 find()

```
auto smtrat::StaticUnionFind::find (
    Value const & val ) -> Representative   [inline], [noexcept]
```

0.15.496.2.2 introduce_variable()

```
void smtrat::StaticUnionFind::introduce_variable (
    Value const & var )  [inline], [noexcept]
```

0.15.496.2.3 merge()

```
void smtrat::StaticUnionFind::merge (
    Value const & a,
    Value const & b )  [inline], [noexcept]
```

0.15.497 smtrat::statistics::StatisticsSettings Struct Reference

`#include <StatisticsSettings.h>`

Data Fields

- bool export_as_xml
- std::string xml_filename
- bool print_as_smtlib

0.15.497.1 Field Documentation

0.15.497.1.1 export_as_xml

`bool smtrat::statistics::StatisticsSettings::export_as_xml`

0.15.497.1.2 print_as_smtlib

`bool smtrat::statistics::StatisticsSettings::print_as_smtlib`

0.15.497.1.3 xml_filename

`std::string smtrat::statistics::StatisticsSettings::xml_filename`

0.15.498 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)

0.15.498.1 Constructor & Destructor Documentation

0.15.498.1.1 StrategyGraph() `smtrat::StrategyGraph::StrategyGraph ( ) [inline]`

0.15.498.2 Member Function Documentation

0.15.498.2.1 addBackend() `template<typename Module >`

```
BackendLink smtrat::StrategyGraph::addBackend (
            const std::initializer_list< BackendLink > & backends ) [inline]
```

0.15.498.2.2 addEdge()

```
BackendLink& smtrat::StrategyGraph::addEdge (
            std::size_t from,
            std::size_t to ) [inline]
```

0.15.498.2.3 addRoot()

```
std::size_t smtrat::StrategyGraph::addRoot (
            const std::initializer_list< BackendLink > & backends ) [inline]
```

0.15.498.2.4 getBackends()

```
std::set<std::pair<thread_priority,AbstractModuleFactory*> > smtrat↩
::StrategyGraph::getBackends (
            std::size_t vertex,
            const carl::Condition & condition ) const [inline]
```

0.15.498.2.5 getRoot()

```
std::size_t smtrat::StrategyGraph::getRoot ( ) const [inline]
```

0.15.498.2.6 hasBranches()

```
bool smtrat::StrategyGraph::hasBranches ( ) const [inline]
```

0.15.498.2.7 numberOfBranches()

```
std::size_t smtrat::StrategyGraph::numberOfBranches ( ) const
[inline]
```

0.15.498.2.8 TrueCondition()

```
static bool smtrat::StrategyGraph::TrueCondition (
            const carl::Condition & c ) [inline], [static]
```

0.15.498.3 Friends And Related Function Documentation

0.15.498.3.1 operator<<

```
std::ostream& operator<< (
            std::ostream & os,
            const StrategyGraph & sg ) [friend]
```

0.15.499 Minisat::StringOption Class Reference

`#include <Options.h>`

Public Member Functions

- StringOption (const char ∗c, const char ∗n, const char ∗d, const char ∗def=NULL)
- operator const char ∗ (void) const
- StringOption & operator= (const char ∗x)
- virtual void help (bool verbose=false)

Static Protected Member Functions

- static vec< Option ∗ > & getOptionList ()
- static const char ∗& getUsageString ()
- static const char ∗& getHelpPrefixString ()

Protected Attributes

- const char ∗ name
- const char ∗ description
- const char ∗ category
- const char ∗ type_name

0.15.499.1 Constructor & Destructor Documentation

0.15.499.1.1 StringOption()

```
Minisat::StringOption::StringOption (
    const char * c,
    const char * n,
    const char * d,
    const char * def = NULL )  [inline]
```

0.15.499.2 Member Function Documentation

0.15.499.2.1 getHelpPrefixString()

`static const char*& Minisat::Option::getHelpPrefixString ( ) [inline], [static], [protected], [inherited]`

0.15.499.2.2 getOptionList()

`static vec<Option*>& Minisat::Option::getOptionList ( ) [inline], [static], [protected], [inherited]`

0.15.499.2.3 getUsageString()

`static const char*& Minisat::Option::getUsageString ( ) [inline], [static], [protected], [inherited]`

0.15.499.2.4 help()

```
virtual void Minisat::StringOption::help (
    bool verbose = false )  [inline], [virtual]
```

Implements Minisat::Option.

0.15.499.2.5 operator const char ∗()

```
Minisat::StringOption::operator const char * (
    void ) const  [inline]
```

0.15.499.2.6 operator=() `StringOption& Minisat::StringOption::operator= ( const char * x ) [inline]`

0.15.499.3 Field Documentation

0.15.499.3.1 category `const char* Minisat::Option::category [protected], [inherited]`

0.15.499.3.2 description `const char* Minisat::Option::description [protected], [inherited]`

0.15.499.3.3 name `const char* Minisat::Option::name [protected], [inherited]`

0.15.499.3.4 type_name `const char* Minisat::Option::type_name [protected], [inherited]`

0.15.500 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

0.15.500.1 Detailed Description

Parses strings: `".+"` with escape sequences `\\"` and `\\\\`

0.15.500.2 Constructor & Destructor Documentation

0.15.500.2.1 StringParser() `smtrat::parser::StringParser::StringParser ( ) [inline]`

0.15.500.3 Field Documentation

0.15.500.3.1 escapes `qi::symbols<char, char> smtrat::parser::StringParser::escapes`

0.15.500.3.2 main `qi::rule<Iterator, std::string(), Skipper> smtrat::parser::StringParser↩ ::main`

0.15.501 smtrat::STropModule< Settings > Class Template Reference

`#include <STropModule.h>`

Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- STropModule (const ModuleInput ∗_formula, Conditionals &_conditionals, Manager ∗_manager=nullptr)
- bool addCore (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- void removeCore (ModuleInput::const_iterator _subformula)

  Removes the subformula of the received formula at the given position to the considered ones of this module.
- void updateModel () const

  Updates the current assignment into the model.
- Answer checkCore ()

  Checks the received formula for consistency.
- bool inform (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void 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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.
- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)
- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.

- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

  Copies the infeasible subsets of the passed formula.
- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

Gets all InformationRelevantFormulas.

- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.501.1 Member Typedef Documentation

0.15.501.1.1 SettingsType

```
template<typename Settings >
typedef Settings smtrat::STropModule< Settings >::SettingsType
```

0.15.501.2 Member Enumeration Documentation

0.15.501.2.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

0.15.501.3 Constructor & Destructor Documentation

0.15.501.3.1 STropModule()

```
template<typename Settings >
smtrat::STropModule< Settings >::STropModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

0.15.501.4 Member Function Documentation

0.15.501.4.1 add()

```
bool smtrat::Module::add (
            ModuleInput::const_iterator _subformula ) [inherited]
```

The module has to take the given sub-formula of the received formula into account.

Parameters

_subformula	The sub-formula to take additionally into account.

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

0.15.501.4.2 addConstraintToInform()

```
void smtrat::Module::addConstraintToInform (
            const FormulaT & _constraint ) [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

0.15.501.4.3 addCore() [1/2]

```
virtual bool smtrat::Module::addCore (
    [[maybe_unused] ] 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.

0.15.501.4.4 addCore() [2/2]

```
template<typename Settings >
bool smtrat::STropModule< Settings >::addCore (
    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.

0.15.501.4.5 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula
(
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.501.4.6 addLemma()

```
void smtrat::Module::addLemma (
    const FormulaT & _lemma,
    const LemmaType & _lt = LemmaType::NORMAL,
    const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline],
[inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.501.4.7 addOrigin()

```
void smtrat::Module::addOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.501.4.8 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.501.4.9 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.501.4.10 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.501.4.11 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.

0.15.501.4.12 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns

All satisfying assignments, if existent.

0.15.501.4.13 anAnswerFound()

```
bool smtrat::Module::anAnswerFound ( ) const [inline], [protected],
[inherited]
```

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.501.4.14 answerFound()

```
const smtrat::Conditionals& smtrat::Module::answerFound ( ) const
[inline], [inherited]
```

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.501.4.15 backendsModel()

```
const Model & smtrat::Module::backendsModel ( ) const [protected],
[inherited]
```

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.501.4.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.501.4.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.501.4.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.501.4.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.501.4.20 check()

```
Answer smtrat::Module::check (
    bool _final = false,
    bool _full = true,
    carl::Variable _objective = carl::Variable::NO_VARIABLE ) [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.501.4.21 checkCore()

```
template<typename Settings >
Answer smtrat::STropModule< Settings >::checkCore ( ) [virtual]
```

Checks the received formula for consistency.

Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; UNKNOWN, otherwise.

Reimplemented from smtrat::Module.

0.15.501.4.22 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.

0.15.501.4.23 checkModel()

```
unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]
```

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.501.4.24 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.501.4.25 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

0.15.501.4.26 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

0.15.501.4.27 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

0.15.501.4.28 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.501.4.29 collectOrigins() [1/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

0.15.501.4.30 collectOrigins() [2/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inherited]`

Collects the formulas in the given formula, which are part of the received formula.
If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.501.4.31 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

0.15.501.4.32 constraintsToInform() `const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns

The constraints which the backends must be informed about.

0.15.501.4.33 currentlySatisfied()

`virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.501.4.34 currentlySatisfiedByBackend()

`unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.501.4.35 deinform()

`void smtrat::Module::deinform ( const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.501.4.36 deinformCore()

`virtual void smtrat::Module::deinformCore ( [[maybe_unused] ] 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.

0.15.501.4.37 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

0.15.501.4.38 eraseSubformulaFromPassedFormula()

```
ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (
            ModuleInput::iterator _subformula,
            bool _ignoreOrigins = false ) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_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 >.

0.15.501.4.39 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

0.15.501.4.40 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
            const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

0.15.501.4.41 firstSubformulaToPass()

```
ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]
```

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.501.4.42 firstUncheckedReceivedSubformula()

```
ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]
```

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.501.4.43 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.501.4.44 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasible↩Subset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.501.4.45 getBackendsAllModels()

`void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.501.4.46 getBackendsModel()

`void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.501.4.47 getInfeasibleSubsets() [1/2]

`void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.501.4.48 getInfeasibleSubsets() [2/2]

`std::vector< FormulaSetT > smtrat::Module::get↩InfeasibleSubsets ( 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.

0.15.501.4.49 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module↩::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.501.4.50 getModelEqualities()

`std::list< std::vector< carl::Variable > > smtrat::Module↩::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.501.4.51 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
const FormulaT & _formula,
FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.501.4.52 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
const FormulaT & _formula,
FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.501.4.53 getOrigins() [3/3]

```
const FormulaT& smtrat::Module::getOrigins (
ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]
```

Gets the origins of the passed formula at the given position.

Parameters

_formula	The position of a formula in the passed formulas.

Returns

The origins of the passed formula at the given position.

0.15.501.4.54 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.

0.15.501.4.55 hasLemmas()

```
bool smtrat::Module::hasLemmas ( ) [inline], [inherited]
```

Checks whether this module or any of its backends provides any lemmas.

0.15.501.4.56 hasValidInfeasibleSubset()

```
bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]
```

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.501.4.57 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.501.4.58 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasible↩ Subsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.501.4.59 inform() `bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.
It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

Parameters

_constraint	The constraint to inform about.

Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

0.15.501.4.60 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.

0.15.501.4.61 informCore() `virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.501.4.62 informedConstraints()

`const carl::FastSet<FormulaT>& smtrat::Module::informedConstraints ( ) const [inline], [inherited]`

Returns

The position of the first constraint of which no backend has been informed about.

0.15.501.4.63 init()

`void smtrat::Module::init ( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not be called twice and only before the first runBackends call.
Reimplemented in smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntBlastModule< Settings >, smtrat::FPPModule< Settings >, smtrat::CurryModule< Settings >, and smtrat::BVModule< Settings >.

0.15.501.4.64 is_minimizing()

`bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.501.4.65 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.

0.15.501.4.66 isPreprocessor()

`bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.501.4.67 lemmas()

`const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.501.4.68 merge()

`std::vector< FormulaT > smtrat::Module::merge ( const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.
({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

Parameters

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns

The vector being the two given vectors merged.

0.15.501.4.69 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.501.4.70 modelsDisjoint() `bool smtrat::Module::modelsDisjoint ( const Model & _modelA, const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

Parameters

_modelA	The first model to check for.
_modelB	The second model to check for.

Returns

true, if there is no variable assigned by both models; false, otherwise.

0.15.501.4.71 moduleName() `template<typename Settings > std::string smtrat::STropModule< Settings >::moduleName ( ) const [inline], [virtual]`

Returns

The name of the given module type as name.

Reimplemented from smtrat::Module.

0.15.501.4.72 objective() `carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

0.15.501.4.73 originInReceivedFormula() `bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.501.4.74 passedFormulaBegin() `ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.501.4.75 passedFormulaEnd() `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.501.4.76 pPassedFormula() `const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.501.4.77 pReceivedFormula() `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.501.4.78 print() `void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.501.4.79 printAllModels() `void smtrat::Module::printAllModels ( std::ostream & _out = std::cout ) [inherited]`

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.501.4.80 printInfeasibleSubsets() `void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.501.4.81 printModel() `void smtrat::Module::printModel ( std::ostream & _out = std::cout ) const [inherited]`

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.501.4.82 printPassedFormula()

`void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.501.4.83 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.501.4.84 probablyLooping()

`bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.501.4.85 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.501.4.86 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

0.15.501.4.87 receivedVariable()

`bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

0.15.501.4.88 remove()

`void smtrat::Module::remove ( ModuleInput::const_iterator _subformula ) [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.
However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

Parameters

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

0.15.501.4.89 removeCore() [1/2]

```
virtual void smtrat::Module::removeCore (
            [[maybe_unused] ] 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.

0.15.501.4.90 removeCore() [2/2]

```
template<typename Settings >
void smtrat::STropModule< Settings >::removeCore (
            ModuleInput::const_iterator _subformula )
```

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.

0.15.501.4.91 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origin (
            ModuleInput::iterator _formula,
            const FormulaT & _origin ) [inline], [protected], [inherited]
```

0.15.501.4.92 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

0.15.501.4.93 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.501.4.94 rReceivedFormula() `const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]`

Returns

A reference to the formula passed to this module.

0.15.501.4.95 runBackends() [1/2] `virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]`

Reimplemented in smtrat::PModule.

0.15.501.4.96 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.

0.15.501.4.97 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.

0.15.501.4.98 setThreadPriority() `void smtrat::Module::setThreadPriority ( thread_priority _threadPriority ) [inline], [inherited]`

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.501.4.99 solverState() `Answer smtrat::Module::solverState ( ) const [inline], [inherited]`

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.501.4.100 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 !=.

0.15.501.4.101 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.501.4.102 updateAllModels()

`void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.501.4.103 updateLemmas()

`void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.501.4.104 updateModel()

`template<typename Settings > void smtrat::STropModule< Settings >::updateModel ( ) const [virtual]`

Updates the current assignment into the model.
Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.
Reimplemented from smtrat::Module.

0.15.501.4.105 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).

0.15.501.5 Field Documentation

0.15.501.5.1 mAllBackends

`std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.501.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.501.5.3 mBackendsFoundAnswer `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.501.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.501.5.5 mFinalCheck `bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.501.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.501.5.7 mFirstSubformulaToPass `ModuleInput::iterator smtrat::Module::mFirstSubformulaTo↩ Pass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.501.5.8 mFirstUncheckedReceivedSubformula `ModuleInput::const_iterator smtrat::Module↩ ::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.501.5.9 mFoundAnswer `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.501.5.10 mFullCheck `bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.501.5.11 mInfeasibleSubsets `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.501.5.12 mInformedConstraints `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.501.5.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.501.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.501.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.501.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.501.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.501.5.18 mObjectiveVariable

`carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.501.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]`

Reusable splitting variables.

0.15.501.5.20 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.501.5.21 mSmallerMusesCheckCounter

`unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.501.5.22 mSolverState

`std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.501.5.23 mTheoryPropagations

`std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]`

0.15.501.5.24 mUsedBackends `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.501.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.502 benchmax::slurm::SubmitfileProperties Struct Reference

All properties needed to create a submit file.

`#include <SlurmUtilities.h>`

Data Fields

- std::string file_suffix

  Suffix for job and submit file.
- std::string filename_jobs

  Filename of the job list file.
- std::string tmp_dir

  Temporary directory for log files.
- carl::settings::duration limit_time

  Time limit in seconds.
- carl::settings::duration grace_time

  Grace time in seconds.
- carl::settings::binary_quantity limit_memory

  Memory limit in megabytes.
- std::size_t tasks

  Number of tasks.
- std::size_t slices

  Number of slices.

0.15.502.1 Detailed Description

All properties needed to create a submit file.

0.15.502.2 Field Documentation

0.15.502.2.1 file_suffix `std::string benchmax::slurm::SubmitfileProperties::file_suffix`

Suffix for job and submit file.

0.15.502.2.2 filename_jobs `std::string benchmax::slurm::SubmitfileProperties::filename_jobs`

Filename of the job list file.

0.15.502.2.3 grace_time `carl::settings::duration benchmax::slurm::SubmitfileProperties::grace↩_time`

Grace time in seconds.

0.15.502.2.4 limit_memory `carl::settings::binary_quantity benchmax::slurm::SubmitfileProperties::limit_memory`

Memory limit in megabytes.

0.15.502.2.5 limit_time `carl::settings::duration benchmax::slurm::SubmitfileProperties::limit_time`

Time limit in seconds.

0.15.502.2.6 slices `std::size_t benchmax::slurm::SubmitfileProperties::slices`

Number of slices.

0.15.502.2.7 tasks `std::size_t benchmax::slurm::SubmitfileProperties::tasks`

Number of tasks.

0.15.502.2.8 tmp_dir `std::string benchmax::slurm::SubmitfileProperties::tmp_dir`

Temporary directory for log files.

0.15.503 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 > &&_oConditions, 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.

0.15.503.1 Member Enumeration Documentation

0.15.503.1.1 Type `enum smtrat::vs::Substitution::Type`

The type of a substitution.

Enumerator

NORMAL	
PLUS_EPSILON	
MINUS_INFINITY	
PLUS_INFINITY	
INVALID	

0.15.503.2 Constructor & Destructor Documentation

0.15.503.2.1 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.

0.15.503.2.2 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.

0.15.503.2.3 Substitution() [3/3]

```
smtrat::vs::Substitution::Substitution (
    const Substitution & _substitution )
```

Copy constructor.

0.15.503.2.4 ~Substitution()

```
smtrat::vs::Substitution::~Substitution ( )
```

The destructor.

0.15.503.3 Member Function Documentation

0.15.503.3.1 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.

0.15.503.3.2 originalConditions()

```
const carl::PointerSet<Condition>& smtrat::vs::Substitution↩
::originalConditions ( ) const [inline]
```

Returns

A constant reference to the original conditions of this substitution.

0.15.503.3.3 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.

0.15.503.3.4 rOriginalConditions() `carl::PointerSet<Condition>& smtrat::vs::Substitution::rOriginalConditions ( ) [inline]`

Returns

A reference to the original conditions of this substitution.

0.15.503.3.5 rType() `Type& smtrat::vs::Substitution::rType ( ) [inline]`

Returns

A reference to the type of this substitution.

0.15.503.3.6 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.

0.15.503.3.7 sideCondition() `const smtrat::ConstraintsT& smtrat::vs::Substitution::sideCondition ( ) const [inline]`

Returns

A constant reference to the side condition of this substitution.

0.15.503.3.8 term() `const smtrat::SqrtEx& smtrat::vs::Substitution::term ( ) const [inline]`

Returns

A constant reference to the term this substitution maps its variable to.

0.15.503.3.9 termVariables() `const carl::Variables& smtrat::vs::Substitution::termVariables ( ) const [inline]`

Returns

A constant reference to the variables occurring in the substitution term.

0.15.503.3.10 type() `const Type& smtrat::vs::Substitution::type ( ) const [inline]`

Returns

A constant reference to the type of this substitution.

0.15.503.3.11 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.

0.15.503.3.12 variable() `const carl::Variable& smtrat::vs::Substitution::variable ( ) const [inline]`

Returns

A constant reference to the variable this substitution substitutes.

0.15.504 smtrat::vs::substitutionPointerEqual Struct Reference

`#include <Substitution.h>`

Public Member Functions

- bool operator() (const Substitution *_substitutionA, const Substitution *_substitutionB) const

0.15.504.1 Member Function Documentation

0.15.504.1.1 operator()() `bool smtrat::vs::substitutionPointerEqual::operator() ( const Substitution * _substitutionA, const Substitution * _substitutionB ) const [inline]`

0.15.505 smtrat::vs::substitutionPointerHash Struct Reference

`#include <Substitution.h>`

Public Member Functions

- size_t operator() (const Substitution *_substitution) const

0.15.505.1 Member Function Documentation

0.15.505.1.1 operator()() `size_t smtrat::vs::substitutionPointerHash::operator() ( const Substitution * _substitution ) const [inline]`

0.15.506 smtrat::cadcells::datastructures::SymbolicInterval Class Reference

A symbolic interal whose bounds are defined by indexed root expressions.

`#include <roots.h>`

Public Member Functions

- SymbolicInterval (IndexedRoot root)

  Constructs a section.
- SymbolicInterval (Bound lower, Bound upper)

  Constructs an interval with arbitrary bounds.
- SymbolicInterval ()

  Constructs (-oo, oo).
- bool is_section () const
- const IndexedRoot & section_defining () const

  In case of a section, the defining indexed root is returned.
- const auto & lower () const

  Return the lower bound.
- const auto & upper () const

  Returns the upper bound.
- void polys (boost::container::flat_set< PolyRef > &result) const
- boost::container::flat_set< PolyRef > polys () const
- void set_to_closure ()

0.15.506.1 Detailed Description

A symbolic interal whose bounds are defined by indexed root expressions.
Bounds can be infty, strict or weak. A special case is a section where the lower and upper bound are equal and weak.

0.15.506.2 Constructor & Destructor Documentation

0.15.506.2.1 SymbolicInterval() [1/3]

```
smtrat::cadcells::datastructures::SymbolicInterval::↩
SymbolicInterval (
            IndexedRoot root ) [inline]
```

Constructs a section.

0.15.506.2.2 SymbolicInterval() [2/3]

```
smtrat::cadcells::datastructures::SymbolicInterval::↩
SymbolicInterval (
            Bound lower,
            Bound upper ) [inline]
```

Constructs an interval with arbitrary bounds.

0.15.506.2.3 SymbolicInterval() [3/3]

```
smtrat::cadcells::datastructures::SymbolicInterval::↩
SymbolicInterval ( ) [inline]
```

Constructs (-oo, oo).

0.15.506.3 Member Function Documentation

0.15.506.3.1 is_section()

```
bool smtrat::cadcells::datastructures::SymbolicInterval::is_section (
) const [inline]
```

0.15.506.3.2 lower() `const auto& smtrat::cadcells::datastructures::SymbolicInterval::lower ( ) const [inline]`

Return the lower bound.

0.15.506.3.3 polys() [1/2] `boost::container::flat_set<PolyRef> smtrat::cadcells::datastructures::SymbolicInterval::polys ( ) const [inline]`

0.15.506.3.4 polys() [2/2] `void smtrat::cadcells::datastructures::SymbolicInterval::polys ( boost::container::flat_set< PolyRef > & result ) const [inline]`

0.15.506.3.5 section_defining() `const IndexedRoot& smtrat::cadcells::datastructures::SymbolicInterval::section_defining ( ) const [inline]`

In case of a section, the defining indexed root is returned.

0.15.506.3.6 set_to_closure() `void smtrat::cadcells::datastructures::SymbolicInterval::set_to_closure ( ) [inline]`

0.15.506.3.7 upper() `const auto& smtrat::cadcells::datastructures::SymbolicInterval::upper ( ) const [inline]`

Returns the upper bound.

0.15.507 smtrat::parser::SymbolParser Struct Reference

`#include <Lexicon.h>`

Public Member Functions

- SymbolParser ()

Data Fields

- qi::rule< Iterator, std::string(), Skipper > main
- qi::rule< Iterator, std::string(), Skipper > quoted
- SimpleSymbolParser simple

0.15.507.1 Constructor & Destructor Documentation

0.15.507.1.1 SymbolParser() `smtrat::parser::SymbolParser::SymbolParser ( ) [inline]`

0.15.507.2 Field Documentation

0.15.507.2.1 main `qi::rule<Iterator, std::string(), Skipper> smtrat::parser::SymbolParser::main`

0.15.507.2.2 quoted `qi::rule<Iterator, std::string(), Skipper> smtrat::parser::SymbolParser::quoted`

0.15.507.2.3 simple `SimpleSymbolParser smtrat::parser::SymbolParser::simple`

0.15.508 smtrat::SymmetryModule< Settings > Class Template Reference

`#include <SymmetryModule.h>`

Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- SymmetryModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=nullptr)
- ~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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.
- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)
- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.

- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

  Copies the infeasible subsets of the passed formula.
- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.508.1 Member Typedef Documentation

0.15.508.1.1 SettingsType `template<typename Settings >`

`typedef Settings smtrat::SymmetryModule< Settings >::SettingsType`

0.15.508.2 Member Enumeration Documentation

0.15.508.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

0.15.508.3 Constructor & Destructor Documentation

0.15.508.3.1 SymmetryModule() `template<typename Settings >`

```
smtrat::SymmetryModule< Settings >::SymmetryModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

0.15.508.3.2 ~SymmetryModule() `template<typename Settings >`

`smtrat::SymmetryModule< Settings >::~SymmetryModule ( )`

0.15.508.4 Member Function Documentation

0.15.508.4.1 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.

0.15.508.4.2 addConstraintToInform() `void smtrat::Module::addConstraintToInform (`

`const FormulaT & _constraint ) [protected], [virtual], [inherited]`

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

0.15.508.4.3 addCore()

```
virtual bool smtrat::Module::addCore (
    [[maybe_unused] ] 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.

0.15.508.4.4 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula
(
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.508.4.5 addLemma()

```
void smtrat::Module::addLemma (
    const FormulaT & _lemma,
    const LemmaType & _lt = LemmaType::NORMAL,
    const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline],
[inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.508.4.6 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.

Parameters

_origin	A set of formulas in the received formula of this module.

0.15.508.4.7 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
            ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.508.4.8 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.508.4.9 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula,
            const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.508.4.10 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.

0.15.508.4.11 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns

All satisfying assignments, if existent.

0.15.508.4.12 anAnswerFound()

```
bool smtrat::Module::anAnswerFound ( ) const [inline], [protected],
[inherited]
```

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.508.4.13 answerFound()

```
const smtrat::Conditionals& smtrat::Module::answerFound ( ) const
[inline], [inherited]
```

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.508.4.14 appliedPreprocessing()

```
bool smtrat::PModule::appliedPreprocessing ( ) const [inline],
[inherited]
```

Returns

true, if the current received formula has been simplified and the result of this simplification is stored in the passed formula.

0.15.508.4.15 backendsModel()

```
const Model & smtrat::Module::backendsModel ( ) const [protected],
[inherited]
```

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.508.4.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline],
[protected], [inherited]
```

0.15.508.4.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.508.4.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline],
[protected], [inherited]
```

0.15.508.4.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.508.4.20 check()

```
Answer smtrat::PModule::check (
    bool _final = false,
    bool _full = true,
    carl::Variable _objective = carl::Variable::NO_VARIABLE ) [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.508.4.21 checkCore()

```
template<typename 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.

0.15.508.4.22 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.

0.15.508.4.23 checkModel()

```
unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]
```

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.508.4.24 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.508.4.25 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

0.15.508.4.26 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

0.15.508.4.27 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

0.15.508.4.28 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.508.4.29 collectOrigins() [1/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

0.15.508.4.30 collectOrigins() [2/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inherited]`

Collects the formulas in the given formula, which are part of the received formula.
If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.508.4.31 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

0.15.508.4.32 constraintsToInform() `const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns

The constraints which the backends must be informed about.

0.15.508.4.33 currentlySatisfied()

`virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.508.4.34 currentlySatisfiedByBackend()

`unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.508.4.35 deinform()

`void smtrat::Module::deinform ( const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.508.4.36 deinformCore()

`virtual void smtrat::Module::deinformCore ( [[maybe_unused] ] 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.

0.15.508.4.37 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

0.15.508.4.38 eraseSubformulaFromPassedFormula()

```
ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (
            ModuleInput::iterator _subformula,
            bool _ignoreOrigins = false ) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_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 >.

0.15.508.4.39 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

0.15.508.4.40 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
            const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

0.15.508.4.41 firstSubformulaToPass()

```
ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]
```

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.508.4.42 firstUncheckedReceivedSubformula()

```
ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]
```

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.508.4.43 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.508.4.44 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasible↩Subset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.508.4.45 getBackendsAllModels()

`void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.508.4.46 getBackendsModel()

`void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.508.4.47 getInfeasibleSubsets() [1/2]

`void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.508.4.48 getInfeasibleSubsets() [2/2]

`std::vector< FormulaSetT > smtrat::Module::get↩InfeasibleSubsets ( 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.

0.15.508.4.49 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module↩::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.508.4.50 getModelEqualities()

`std::list< std::vector< carl::Variable > > smtrat::Module↩::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.508.4.51 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
const FormulaT & _formula,
FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.508.4.52 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
const FormulaT & _formula,
FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.508.4.53 getOrigins() [3/3]

```
const FormulaT& smtrat::Module::getOrigins (
ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]
```

Gets the origins of the passed formula at the given position.

Parameters

_formula	The position of a formula in the passed formulas.

Returns

The origins of the passed formula at the given position.

0.15.508.4.54 getReceivedFormulaSimplified()

```
std::pair< bool, FormulaT > smtrat::PModule::getReceivedFormulaSimplified ( ) [virtual], [inherited]
```

Returns

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's reveived formula, if the Boolean is true.

Reimplemented from smtrat::Module.

0.15.508.4.55 hasLemmas()

```
bool smtrat::Module::hasLemmas ( ) [inline], [inherited]
```

Checks whether this module or any of its backends provides any lemmas.

0.15.508.4.56 hasValidInfeasibleSubset()

```
bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]
```

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.508.4.57 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.508.4.58 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasible↩ Subsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.508.4.59 inform() `bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.
It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

Parameters

_constraint	The constraint to inform about.

Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

0.15.508.4.60 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.

0.15.508.4.61 informCore() `virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.508.4.62 informedConstraints()

`const carl::FastSet<FormulaT>& smtrat::Module::informedConstraints ( ) const [inline], [inherited]`

Returns

The position of the first constraint of which no backend has been informed about.

0.15.508.4.63 init()

`void smtrat::Module::init ( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not be called twice and only before the first runBackends call.
Reimplemented in smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntBlastModule< Settings >, smtrat::FPPModule< Settings >, smtrat::CurryModule< Settings >, and smtrat::BVModule< Settings >.

0.15.508.4.64 is_minimizing()

`bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.508.4.65 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.

0.15.508.4.66 isPreprocessor()

`bool smtrat::PModule::isPreprocessor ( ) const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.508.4.67 lemmas()

`const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.508.4.68 merge()

`std::vector< FormulaT > smtrat::Module::merge ( const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.
({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

Parameters

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns

The vector being the two given vectors merged.

0.15.508.4.69 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.508.4.70 modelsDisjoint() `bool smtrat::Module::modelsDisjoint ( const Model & _modelA, const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

Parameters

_modelA	The first model to check for.
_modelB	The second model to check for.

Returns

true, if there is no variable assigned by both models; false, otherwise.

0.15.508.4.71 moduleName() `virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]`

Returns

The name of the given module type as name.

Reimplemented in smtrat::VSModule< Settings >, smtrat::UnionFindModule< Settings >, smtrat::UFCegarModule< Settings >, smtrat::STropModule< Settings >, smtrat::SplitSOSModule< Settings >, smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LVEModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntEqModule< Settings >, smtrat::IntBlastModule< Settings >, smtrat::IncWidthModule< Settings >, smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, smtrat::ICPModule< ICPSettings1 >, smtrat::GBModule< Settings >, smtrat::FouMoModule< Settings >, smtrat::CurryModule< Settings >, smtrat::CubeLIAModule< Settings >, smtrat::CSplitModule< Settings >, smtrat::BVModule< Settings >, and smtrat::BEModule< Settings >.

0.15.508.4.72 objective() `carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

0.15.508.4.73 originInReceivedFormula() `bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.508.4.74 passedFormulaBegin() `ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.508.4.75 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.508.4.76 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.508.4.77 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.508.4.78 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.508.4.79 printAllModels()

`void smtrat::Module::printAllModels ( std::ostream & _out = std::cout ) [inherited]`

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.508.4.80 printInfeasibleSubsets()

`void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.508.4.81 printModel()

`void smtrat::Module::printModel ( std::ostream & _out = std::cout ) const [inherited]`

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.508.4.82 printPassedFormula()

`void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.508.4.83 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.508.4.84 probablyLooping()

`bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.508.4.85 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.508.4.86 receivedFormulasAsInfeasibleSubset()

`void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

0.15.508.4.87 receivedVariable()

```
bool smtrat::Module::receivedVariable (
    carl::Variable::Arg _var ) const [inline], [inherited]
```

0.15.508.4.88 remove()

```
void smtrat::PModule::remove (
    ModuleInput::const_iterator _subformula ) [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.
However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

Parameters

_subformula	The sub formula of the received formula to remove.

Reimplemented from smtrat::Module.

0.15.508.4.89 removeCore()

```
virtual void smtrat::Module::removeCore (
    [[maybe_unused] ] 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.

0.15.508.4.90 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

0.15.508.4.91 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
    ModuleInput::iterator _formula,
    const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],
[inherited]
```

0.15.508.4.92 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.508.4.93 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to this module.

0.15.508.4.94 runBackends() [1/2]

`virtual Answer smtrat::PModule::runBackends ( ) [inline], [virtual], [inherited]`

Reimplemented from smtrat::Module.

0.15.508.4.95 runBackends() [2/2]

```
Answer smtrat::PModule::runBackends (
    bool _final,
    bool _full,
    carl::Variable _objective ) [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

Parameters

_final	true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT.
_full	false, if this module should avoid too expensive procedures and rather return unknown instead.
_objective	if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good.

Returns

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.508.4.96 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.

0.15.508.4.97 setThreadPriority()

```
void smtrat::Module::setThreadPriority (
    thread_priority _threadPriority ) [inline], [inherited]
```

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.508.4.98 solverState()

`Answer smtrat::Module::solverState ( ) const [inline], [inherited]`

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.508.4.99 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 !=.

0.15.508.4.100 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.508.4.101 updateAllModels()

`void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.508.4.102 updateLemmas()

`void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.508.4.103 updateModel()

`void smtrat::PModule::updateModel ( ) const [virtual], [inherited]`

Updates the current assignment into the model.
Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.
Reimplemented from smtrat::Module.

0.15.508.4.104 usedBackends()

`const std::vector<Module*>& smtrat::Module::usedBackends ( ) const [inline], [inherited]`

Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.508.5 Field Documentation

0.15.508.5.1 mAllBackends

`std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.508.5.2 mAllModels

`std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.508.5.3 mBackendsFoundAnswer `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.508.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.508.5.5 mFinalCheck `bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.508.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.508.5.7 mFirstSubformulaToPass `ModuleInput::iterator smtrat::Module::mFirstSubformulaTo↩ Pass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.508.5.8 mFirstUncheckedReceivedSubformula `ModuleInput::const_iterator smtrat::Module↩ ::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.508.5.9 mFoundAnswer `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.508.5.10 mFullCheck `bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.508.5.11 mInfeasibleSubsets `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.508.5.12 mInformedConstraints `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.508.5.13 mLastBranches `std::vector< Branching > smtrat::Module::mLastBranches = std↩ ::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.508.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.508.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.508.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.508.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.508.5.18 mObjectiveVariable

`carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.508.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplitting↩ Variables [static], [inherited]`

Reusable splitting variables.

0.15.508.5.20 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.508.5.21 mSmallerMusesCheckCounter

`unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.508.5.22 mSolverState

`std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.508.5.23 mTheoryPropagations

`std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

0.15.508.5.24 mUsedBackends

`std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.508.5.25 mVariableCounters

`std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.509 smtrat::lra::Tableau< Settings, T1, T2 > Class Template Reference

`#include <Tableau.h>`

Data Structures

- struct LearnedBound

Public Member Functions

- Tableau (ModuleInput::iterator _defaultBoundPosition)
- ~Tableau ()
- void setSize (size_t _expectedHeight, size_t _expectedWidth, size_t _expectedNumberOfBounds)
- size_t size () const
- void setBlandsRuleStart (size_t _start)
- const std::vector< Variable< T1, T2 > * > & rows () const
- const std::vector< Variable< T1, T2 > * > & columns () const
- const carl::FastMap< carl::Variable, Variable< T1, T2 > * > & originalVars () const
- const carl::FastPointerMap< typename Poly::PolyType, Variable< T1, T2 > * > & slackVars () const
- const T1 & currentDelta () const
- const carl::FastMap< FormulaT, std::vector< const Bound< T1, T2 > * > * > & constraintToBound () const
- carl::FastMap< FormulaT, std::vector< const Bound< T1, T2 > * > * > & rConstraintToBound ()
- size_t numberOfPivotingSteps () const
- void resetNumberOfPivotingSteps ()
- carl::FastPointerMap< Variable< T1, T2 >, LearnedBound > & rLearnedLowerBounds ()
- carl::FastPointerMap< Variable< T1, T2 >, LearnedBound > & rLearnedUpperBounds ()
- void resetTheta ()
- std::vector< typename carl::FastPointerMap< Variable< T1, T2 >, LearnedBound >::iterator > & rNewLearnedBounds ()
- bool entryIsPositive (const TableauEntry< T1, T2 > &_entry) const
- bool entryIsNegative (const TableauEntry< T1, T2 > &_entry) const
- auto defaultBoundPosition () const
- bool isActive (const Variable< T1, T2 > &_var) const
- EntryID newTableauEntry (const T2 &_content)
- void removeEntry (EntryID _entryID)
- Variable< T1, T2 > * getVariable (const Poly &_lhs, T1 &_factor, T1 &_boundValue)
- Variable< T1, T2 > * getObjectiveVariable (const Poly &_lhs)
- std::pair< const Bound< T1, T2 > *, bool > newBound (const FormulaT &_constraint)
- void removeBound (const FormulaT &_constraint)
- void activateBound (const Bound< T1, T2 > *_bound, const FormulaT &_formula)
- void deleteVariable (Variable< T1, T2 > *_variable, bool _optimizationVar=false)
- Variable< T1, T2 > * newNonbasicVariable (const typename Poly::PolyType *_poly, bool _isInteger)
- Variable< T1, T2 > * newBasicVariable (const typename Poly::PolyType *_poly, bool _isInteger)
- void activateBasicVar (Variable< T1, T2 > *_var)

  g
- void deactivateBasicVar (Variable< T1, T2 > *_var)
- void storeAssignment ()
- void resetAssignment ()
- RationalAssignment getRationalAssignment () const
- void adaptDelta (const Variable< T1, T2 > &_variable, bool _upperBound, T1 &_minDelta) const
- void compressRows ()
- bool usedBlandsRule ()

  Returns true, if the next pivoting element is selected by blands rule.
- bool hasMultilineConflict ()
- std::vector< const Bound< T1, T2 > * > getMultilineConflict ()
- std::pair< EntryID, bool > nextPivotingElement ()

- Value< T1 > violationSum ()

  Method to compute the sum of derivations to the closest bounds.
- Value< T1 > dViolationSum (const Variable< T1, T2 > *nVar, const Value< T1 > &update)

  Computes the change in the violation sum when an the update value is added to the assignment of the 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 std::size_t i, std::vector< std::pair< Value< T1 >, Variable< T1, T2 > * > > &leaving_candidates)

  Helper method to compute leavind candidates used in nextPivotingElementInfeasibilities.
- void updateNonbasicVariable (EntryID _pivotingElement)

  Updates a nonbasic variable by the assignment stored in mpTheta and triggers updates of depending basic variables.
- void updateNonbasicVariable (EntryID _pivotingElement, Value< T1 > update)

  Updates a nonbasic variable by the given assignment and triggers updates of depending basic variables.
- std::pair< EntryID, bool > hasConflict ()

  Checks if a conflicting pair exists.
- void printEntries ()
- std::vector< T2 > getInfeasibilityRow ()

  Method to compute the infeasibility row to the current tableau.
- void setInfeasibilityRow ()

  Method to set the 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

0.15.509.1 Constructor & Destructor Documentation

0.15.509.1.1 Tableau()

```
template<class Settings , typename T1 , typename T2 >
smtrat::lra::Tableau< Settings, T1, T2 >::Tableau (
            ModuleInput::iterator _defaultBoundPosition )
```

Parameters

_defaultBoundPosition	

0.15.509.1.2 ~Tableau()

```
template<class Settings , typename T1 , typename T2 >
smtrat::lra::Tableau< Settings, T1, T2 >::~Tableau ( )
```

0.15.509.2 Member Function Documentation

0.15.509.2.1 activateBasicVar()

```
template<class Settings , typename T1 , typename T2 >
void smtrat::lra::Tableau< Settings, T1, T2 >::activateBasicVar (
            Variable< T1, T2 > * _var )
```

g

Parameters

_var	

0.15.509.2.2 activateBound()

```
template<class Settings , typename T1 , typename T2 >
void smtrat::lra::Tableau< Settings, T1, T2 >::activateBound (
            const Bound< T1, T2 > * _bound,
            const FormulaT & _formula )
```

Parameters

_bound	
_formula	

0.15.509.2.3 adaptDelta()

```
template<class Settings , typename T1 , typename T2 >
void smtrat::lra::Tableau< Settings, T1, T2 >::adaptDelta (
            const Variable< T1, T2 > & _variable,
            bool _upperBound,
            T1 & _minDelta ) const
```

0.15.509.2.4 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.

0.15.509.2.5 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

0.15.509.2.6 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

0.15.509.2.7 checkCorrectness()

```
template<class Settings , typename T1 , typename T2 >
size_t smtrat::lra::Tableau< Settings, T1, T2 >::checkCorrectness ( ) const
```

Returns

0.15.509.2.8 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	

0.15.509.2.9 columns()

```
template<class Settings , typename T1 , typename T2 >
const std::vector<Variable<T1, T2>*>& smtrat::lra::Tableau< Settings, T1, T2 >::columns ( )
const  [inline]
```

Returns

0.15.509.2.10 compressRows()

```
template<class Settings , typename T1 , typename T2 >
void smtrat::lra::Tableau< Settings, T1, T2 >::compressRows ( )
```

0.15.509.2.11 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

0.15.509.2.12 computeLeavingCandidates()

```
template<class Settings , typename T1 , typename T2 >
void smtrat::lra::Tableau< Settings, T1, T2 >::computeLeavingCandidates (
            const std::size_t i,
            std::vector< std::pair< Value< T1 >, Variable< T1, T2 > * > > & leaving_↩
candidates )
```

Helper method to compute leavind candidates used in nextPivotingElementInfeasibilities.
Computes for every non-basic variable the breaking points. Updates from non-basic vars of 0 are omitted to prevent loops.

Parameters

i	: Index if entering (nonbasic) variable.
leaving_candidates	: Used as storage object for update candidates.

0.15.509.2.13 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

0.15.509.2.14 currentDelta()

```
template<class Settings , typename T1 , typename T2 >
const T1& smtrat::lra::Tableau< Settings, T1, T2 >::currentDelta ( ) const  [inline]
```

0.15.509.2.15 deactivateBasicVar()

```
template<class Settings , typename T1 , typename T2 >
void smtrat::lra::Tableau< Settings, T1, T2 >::deactivateBasicVar (
            Variable< T1, T2 > * _var )
```

Parameters

_var	

0.15.509.2.16 defaultBoundPosition()

```
template<class Settings , typename T1 , typename T2 >
auto smtrat::lra::Tableau< Settings, T1, T2 >::defaultBoundPosition ( ) const  [inline]
```

Returns

0.15.509.2.17 deleteVariable()

```
template<class Settings , typename T1 , typename T2 >
void smtrat::lra::Tableau< Settings, T1, T2 >::deleteVariable (
            Variable< T1, T2 > * _variable,
            bool _optimizationVar = false )
```

Parameters

_variable	

0.15.509.2.18 dViolationSum()

```
template<class Settings , typename T1 , typename T2 >
Value<T1> smtrat::lra::Tableau< Settings, T1, T2 >::dViolationSum (
            const Variable< T1, T2 > * nVar,
            const Value< T1 > & update )
```

Computes the change in the violation sum when an the update value is added to the assignment of the nonbassi variable nVar.

Parameters

nVar	variable to be updated
update	Update value

0.15.509.2.19 entryIsNegative()

```
template<class Settings , typename T1 , typename T2 >
bool smtrat::lra::Tableau< Settings, T1, T2 >::entryIsNegative (
            const TableauEntry< T1, T2 > & _entry ) const  [inline]
```

0.15.509.2.20 entryIsPositive()

```
template<class Settings , typename T1 , typename T2 >
bool smtrat::lra::Tableau< Settings, T1, T2 >::entryIsPositive (
            const TableauEntry< T1, T2 > & _entry ) const  [inline]
```

0.15.509.2.21 get_dVioSum()

```
template<class Settings , typename T1 , typename T2 >
Value<T1> smtrat::lra::Tableau< Settings, T1, T2 >::get_dVioSum ( )  [inline]
```

Used as assertion-function to check the progress of the tableau.

0.15.509.2.22 getConflict()

```
template<class Settings , typename T1 , typename T2 >
std::vector< const Bound<T1, T2>* > smtrat::lra::Tableau< Settings, T1, T2 >::getConflict (
            EntryID _rowEntry ) const
```

Parameters

_rowEntry	

Returns

0.15.509.2.23 getConflictsFrom()

```
template<class Settings , typename T1 , typename T2 >
std::vector< std::vector< const Bound<T1, T2>* > > smtrat::lra::Tableau< Settings, T1, T2
```

```
>::getConflictsFrom (
            EntryID _rowEntry ) const
```

Parameters

_rowEntry	

Returns

0.15.509.2.24 getInfeasibilityRow()

```
template<class Settings , typename T1 , typename T2 >
std::vector<T2> smtrat::lra::Tableau< Settings, T1, T2 >::getInfeasibilityRow ( )
```

Method to compute the infeasibility row to the current tableau.

0.15.509.2.25 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.

0.15.509.2.26 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

0.15.509.2.27 getObjectiveVariable()

```
template<class Settings , typename T1 , typename T2 >
Variable<T1,T2>* smtrat::lra::Tableau< Settings, T1, T2 >::getObjectiveVariable (
            const Poly & _lhs )
```

0.15.509.2.28 getRationalAssignment()

```
template<class Settings , typename T1 , typename T2 >
RationalAssignment smtrat::lra::Tableau< Settings, T1, T2 >::getRationalAssignment ( ) const
```

Returns

0.15.509.2.29 getVariable()

```
template<class Settings , typename T1 , typename T2 >
Variable<T1,T2>* smtrat::lra::Tableau< Settings, T1, T2 >::getVariable (
```

```
            const Poly & _lhs,
            T1 & _factor,
            T1 & _boundValue )
```

0.15.509.2.30 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.

0.15.509.2.31 hasConflict()

```
template<class Settings , typename T1 , typename T2 >
std::pair<EntryID,bool> smtrat::lra::Tableau< Settings, T1, T2 >::hasConflict ( )
```

Checks if a conflicting pair exists.
In a conflict is found, the cell and false is returned. If not, LAST_ENTRY_ID and true is returned.

0.15.509.2.32 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.

0.15.509.2.33 isActive()

```
template<class Settings , typename T1 , typename T2 >
bool smtrat::lra::Tableau< Settings, T1, T2 >::isActive (
            const Variable< T1, T2 > & _var ) const  [inline]
```

0.15.509.2.34 isConflicting()

```
template<class Settings , typename T1 , typename T2 >
bool smtrat::lra::Tableau< Settings, T1, T2 >::isConflicting ( ) const
```

0.15.509.2.35 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

0.15.509.2.36 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

0.15.509.2.37 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

0.15.509.2.38 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

0.15.509.2.39 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

0.15.509.2.40 newTableauEntry()

```
template<class Settings , typename T1 , typename T2 >
EntryID smtrat::lra::Tableau< Settings, T1, T2 >::newTableauEntry (
            const T2 & _content )
```

Parameters

_content	

Returns

0.15.509.2.41 nextPivotingElement()

```
template<class Settings , typename T1 , typename T2 >
std::pair<EntryID, bool> smtrat::lra::Tableau< Settings, T1, T2 >::nextPivotingElement ( )
```

Returns

0.15.509.2.42 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

0.15.509.2.43 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

0.15.509.2.44 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
```

0.15.509.2.45 numberOfPivotingSteps()

```
template<class Settings , typename T1 , typename T2 >
size_t smtrat::lra::Tableau< Settings, T1, T2 >::numberOfPivotingSteps ( ) const  [inline]
```

Returns

0.15.509.2.46 optimizeIndependentNonbasics()

```
template<class Settings , typename T1 , typename T2 >
std::pair<EntryID,bool> smtrat::lra::Tableau< Settings, T1, T2 >::optimizeIndependentNonbasics
(
            const Variable< T1, T2 > & _objective )
```

Parameters

_objective	

Returns

0.15.509.2.47 originalVars()

```
template<class Settings , typename T1 , typename T2 >
const carl::FastMap< carl::Variable, Variable<T1,T2>*>& smtrat::lra::Tableau< Settings, T1,
T2 >::originalVars ( ) const  [inline]
```

Returns

0.15.509.2.48 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

0.15.509.2.49 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	

0.15.509.2.50 printEntries()

```
template<class Settings , typename T1 , typename T2 >
void smtrat::lra::Tableau< Settings, T1, T2 >::printEntries ( ) [inline]
```

0.15.509.2.51 printEntry()

```
template<class Settings , typename T1 , typename T2 >
void smtrat::lra::Tableau< Settings, T1, T2 >::printEntry (
            EntryID _entry,
            std::ostream & _out = std::cout,
            int _maxEntryLength = 20 ) const
```

Parameters

_entry	
_out	
_maxEntryLength	

0.15.509.2.52 printHeap()

```
template<class Settings , typename T1 , typename T2 >
void smtrat::lra::Tableau< Settings, T1, T2 >::printHeap (
            std::ostream & _out = std::cout,
            int _maxEntryLength = 30,
            const std::string _init = "" ) const
```

Parameters

_out	
_maxEntryLength	
_init	

0.15.509.2.53 printLearnedBound()

```
template<class Settings , typename T1 , typename T2 >
void smtrat::lra::Tableau< Settings, T1, T2 >::printLearnedBound (
            const Variable< T1, T2 > & _var,
            const LearnedBound & _learnedBound,
            const std::string _init = "",
            std::ostream & _out = std::cout ) const
```

Parameters

_var	
_learnedBound	
_init	
_out	

0.15.509.2.54 printLearnedBounds()

```
template<class Settings , typename T1 , typename T2 >
void smtrat::lra::Tableau< Settings, T1, T2 >::printLearnedBounds (
            const std::string _init = "",
            std::ostream & _out = std::cout ) const
```

Parameters

_init	
_out	

0.15.509.2.55 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	

0.15.509.2.56 rConstraintToBound()

```
template<class Settings , typename T1 , typename T2 >
carl::FastMap<FormulaT, std::vector<const Bound<T1, T2>*>*>& smtrat::lra::Tableau< Settings,
T1, T2 >::rConstraintToBound ( )  [inline]
```

Returns

0.15.509.2.57 removeBound()

```
template<class Settings , typename T1 , typename T2 >
void smtrat::lra::Tableau< Settings, T1, T2 >::removeBound (
            const FormulaT & _constraint )
```

0.15.509.2.58 removeEntry()

```
template<class Settings , typename T1 , typename T2 >
void smtrat::lra::Tableau< Settings, T1, T2 >::removeEntry (
            EntryID _entryID )
```

Parameters

_entryID	

0.15.509.2.59 resetAssignment()

```
template<class Settings , typename T1 , typename T2 >
void smtrat::lra::Tableau< Settings, T1, T2 >::resetAssignment ( )
```

0.15.509.2.60 resetNumberOfPivotingSteps()

```
template<class Settings , typename T1 , typename T2 >
void smtrat::lra::Tableau< Settings, T1, T2 >::resetNumberOfPivotingSteps ( ) [inline]
```

0.15.509.2.61 resetTheta()

```
template<class Settings , typename T1 , typename T2 >
void smtrat::lra::Tableau< Settings, T1, T2 >::resetTheta ( ) [inline]
```

0.15.509.2.62 rLearnedLowerBounds()

```
template<class Settings , typename T1 , typename T2 >
carl::FastPointerMap<Variable<T1,T2>, LearnedBound>& smtrat::lra::Tableau< Settings, T1, T2 >::rLearnedLowerBounds ( ) [inline]
```

Returns

0.15.509.2.63 rLearnedUpperBounds()

```
template<class Settings , typename T1 , typename T2 >
carl::FastPointerMap<Variable<T1,T2>, LearnedBound>& smtrat::lra::Tableau< Settings, T1, T2 >::rLearnedUpperBounds ( ) [inline]
```

Returns

0.15.509.2.64 rNewLearnedBounds()

```
template<class Settings , typename T1 , typename T2 >
std::vector<typename carl::FastPointerMap<Variable<T1,T2>, LearnedBound>::iterator>& smtrat::lra::Tableau< Settings, T1, T2 >::rNewLearnedBounds ( ) [inline]
```

Returns

0.15.509.2.65 rowCorrect()

```
template<class Settings , typename T1 , typename T2 >
bool smtrat::lra::Tableau< Settings, T1, T2 >::rowCorrect (
            size_t _rowNumber ) const
```

Parameters

_rowNumber	

Returns

0.15.509.2.66 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.

0.15.509.2.67 rows()

```
template<class Settings , typename T1 , typename T2 >
const std::vector<Variable<T1, T2>*>& smtrat::lra::Tableau< Settings, T1, T2 >::rows ( )
const  [inline]
```

Returns

0.15.509.2.68 setBlandsRuleStart()

```
template<class Settings , typename T1 , typename T2 >
void smtrat::lra::Tableau< Settings, T1, T2 >::setBlandsRuleStart (
            size_t _start )  [inline]
```

0.15.509.2.69 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.

0.15.509.2.70 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	

0.15.509.2.71 size()

```
template<class Settings , typename T1 , typename T2 >
size_t smtrat::lra::Tableau< Settings, T1, T2 >::size ( ) const  [inline]
```

Returns

0.15.509.2.72 slackVars()

```
template<class Settings , typename T1 , typename T2 >
const carl::FastPointerMap<typename Poly::PolyType, Variable<T1,T2>*>& smtrat::lra::Tableau<
Settings, T1, T2 >::slackVars ( ) const  [inline]
```

Returns

0.15.509.2.73 storeAssignment()

```
template<class Settings , typename T1 , typename T2 >
void smtrat::lra::Tableau< Settings, T1, T2 >::storeAssignment ( )
```

0.15.509.2.74 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

0.15.509.2.75 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.

0.15.509.2.76 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	

0.15.509.2.77 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

0.15.509.2.78 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

0.15.509.2.79 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.

0.15.509.2.80 violationSum()

```
template<class Settings , typename T1 , typename T2 >
Value<T1> smtrat::lra::Tableau< Settings, T1, T2 >::violationSum ( )
```

Method to compute the sum of derivations to the closest bounds.

0.15.510 smtrat::lra::TableauEntry< T1, T2 > Class Template Reference

```
#include <Tableau.h>
```

Public Member Functions

- TableauEntry ()
- TableauEntry (EntryID _up, EntryID _down, EntryID _left, EntryID _right, Variable< T1, T2 > *_rowVar, Variable< T1, T2 > *_columnVar, const T2 &_content)
- TableauEntry (const TableauEntry &_entry)
- ~TableauEntry ()
- void setVNext (bool downwards, const EntryID _entryId)
- void setHNext (bool leftwards, const EntryID _entryId)

- EntryID vNext (bool downwards)
- EntryID hNext (bool leftwards)
- Variable< T1, T2 > * rowVar () const
- void setRowVar (Variable< T1, T2 > *_rowVar)
- Variable< T1, T2 > * columnVar () const
- void setColumnVar (Variable< T1, T2 > *_columnVar)
- const T2 & content () const
- T2 & rContent ()

0.15.510.1 Constructor & Destructor Documentation

0.15.510.1.1 TableauEntry() [1/3]

```
template<typename T1 , typename T2 >
smtrat::lra::TableauEntry< T1, T2 >::TableauEntry ( ) [inline]
```

0.15.510.1.2 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	

0.15.510.1.3 TableauEntry() [3/3]

```
template<typename T1 , typename T2 >
smtrat::lra::TableauEntry< T1, T2 >::TableauEntry (
            const TableauEntry< T1, T2 > & _entry ) [inline]
```

Parameters

_entry	

0.15.510.1.4 ~TableauEntry()

```
template<typename T1 , typename T2 >
smtrat::lra::TableauEntry< T1, T2 >::~TableauEntry ( ) [inline]
```

0.15.510.2 Member Function Documentation

0.15.510.2.1 columnVar() `template<typename T1 , typename T2 >`

`Variable<T1, T2>* smtrat::lra::TableauEntry< T1, T2 >::columnVar ( ) const [inline]`

Returns

0.15.510.2.2 content() `template<typename T1 , typename T2 >`

`const T2& smtrat::lra::TableauEntry< T1, T2 >::content ( ) const [inline]`

Returns

0.15.510.2.3 hNext() `template<typename T1 , typename T2 >`

```
EntryID smtrat::lra::TableauEntry< T1, T2 >::hNext (
            bool leftwards ) [inline]
```

Parameters

leftwards	

Returns

0.15.510.2.4 rContent() `template<typename T1 , typename T2 >`

`T2& smtrat::lra::TableauEntry< T1, T2 >::rContent ( ) [inline]`

Returns

0.15.510.2.5 rowVar() `template<typename T1 , typename T2 >`

`Variable<T1, T2>* smtrat::lra::TableauEntry< T1, T2 >::rowVar ( ) const [inline]`

Returns

0.15.510.2.6 setColumnVar() `template<typename T1 , typename T2 >`

```
void smtrat::lra::TableauEntry< T1, T2 >::setColumnVar (
            Variable< T1, T2 > * _columnVar ) [inline]
```

Parameters

_columnVar	

0.15.510.2.7 setHNext() `template<typename T1 , typename T2 >`

```
void smtrat::lra::TableauEntry< T1, T2 >::setHNext (
            bool leftwards,
            const EntryID _entryId )  [inline]
```

Parameters

leftwards	
_entryId	

0.15.510.2.8 setRowVar() template<typename T1 , typename T2 >

```
void smtrat::lra::TableauEntry< T1, T2 >::setRowVar (
            Variable< T1, T2 > * _rowVar )  [inline]
```

Parameters

_rowVar	

0.15.510.2.9 setVNext() template<typename T1 , typename T2 >

```
void smtrat::lra::TableauEntry< T1, T2 >::setVNext (
            bool downwards,
            const EntryID _entryId )  [inline]
```

Parameters

downwards	
_entryId	

0.15.510.2.10 vNext() template<typename T1 , typename T2 >

```
EntryID smtrat::lra::TableauEntry< T1, T2 >::vNext (
            bool downwards )  [inline]
```

Parameters

downwards	

Returns

0.15.511 smtrat::cadcells::datastructures::TaggedIndexedRoot Struct Reference

```
#include <delineation.h>
```

Data Fields

- IndexedRoot root
- bool is_inclusive = false
- bool is_optional = false
- std::optional< PolyRef > origin = std::nullopt

0.15.511.1 Field Documentation

0.15.511.1.1 is_inclusive `bool smtrat::cadcells::datastructures::TaggedIndexedRoot::is_inclusive = false`

0.15.511.1.2 is_optional `bool smtrat::cadcells::datastructures::TaggedIndexedRoot::is_optional = false`

0.15.511.1.3 origin `std::optional<PolyRef> smtrat::cadcells::datastructures::TaggedIndexed↩ Root::origin = std::nullopt`

0.15.511.1.4 root `IndexedRoot smtrat::cadcells::datastructures::TaggedIndexedRoot::root`

0.15.512 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

0.15.512.1 Detailed Description

Tagged Polynomials.

0.15.512.2 Field Documentation

0.15.512.2.1 deg `std::size_t smtrat::mcsat::onecellcad::TagPoly::deg = 0`

0.15.512.2.2 level `std::size_t smtrat::mcsat::onecellcad::TagPoly::level`

0.15.512.2.3 poly `Poly smtrat::mcsat::onecellcad::TagPoly::poly`

0.15.512.2.4 tag `InvarianceType smtrat::mcsat::onecellcad::TagPoly::tag`

0.15.513 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

0.15.513.1 Constructor & Destructor Documentation

0.15.513.1.1 Task() `template<typename T >`

```
smtrat::Task::Task (
            T && task,
            const Module * module,
            std::size_t _index )  [inline]
```

0.15.513.2 Member Function Documentation

0.15.513.2.1 conditionalIndex() `std::size_t smtrat::Task::conditionalIndex ( ) const  [inline]`

0.15.513.2.2 getFuture() `std::future<Answer> smtrat::Task::getFuture ( )  [inline]`

0.15.513.2.3 getModule() `const Module* smtrat::Task::getModule ( ) const  [inline]`

0.15.513.2.4 operator<() `bool smtrat::Task::operator< (`

```
            const Task & rhs ) const
```

0.15.513.2.5 run() `void smtrat::Task::run ( )  [inline]`

0.15.514 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

0.15.514.1 Member Typedef Documentation

0.15.514.1.1 Converter `typedef conversion::VariantVariantConverter<types::TermType> smtrat::parser::TermParse`

0.15.514.2 Constructor & Destructor Documentation

0.15.514.2.1 TermParser() `smtrat::parser::TermParser::TermParser ( Theories * theories ) [inline]`

0.15.514.3 Field Documentation

0.15.514.3.1 attribute `AttributeParser smtrat::parser::TermParser::attribute`

0.15.514.3.2 binding `qi::rule<Iterator, Skipper> smtrat::parser::TermParser::binding`

0.15.514.3.3 converter `Converter smtrat::parser::TermParser::converter`

0.15.514.3.4 main `qi::rule<Iterator, types::TermType(), Skipper> smtrat::parser::TermParser↩ ::main`

0.15.514.3.5 qualifiedidentifier `QualifiedIdentifierParser smtrat::parser::TermParser::qualifiedidentifier`

0.15.514.3.6 sortedvariable `SortedVariableParser smtrat::parser::TermParser::sortedvariable`

0.15.514.3.7 specconstant `SpecConstantParser smtrat::parser::TermParser::specconstant`

0.15.514.3.8 symbol `SymbolParser smtrat::parser::TermParser::symbol`

0.15.514.3.9 termop `qi::rule<Iterator, types::TermType(), Skipper> smtrat::parser::TermParser::termop`

0.15.514.3.10 theories `Theories* smtrat::parser::TermParser::theories`

0.15.515 smtrat::mcsat::vs::helper::TestCandidate Struct Reference

`#include <VSHelper.h>`

Public Member Functions

- bool operator== (const TestCandidate &other) const

Data Fields

- carl::vs::Term< Poly > term
- ConstraintsT side_condition

0.15.515.1 Member Function Documentation

0.15.515.1.1 operator==() `bool smtrat::mcsat::vs::helper::TestCandidate::operator== ( const TestCandidate & other ) const [inline]`

0.15.515.2 Field Documentation

0.15.515.2.1 side_condition `ConstraintsT smtrat::mcsat::vs::helper::TestCandidate::side_condition`

0.15.515.2.2 term `carl::vs::Term<Poly> smtrat::mcsat::vs::helper::TestCandidate::term`

0.15.516 smtrat::parser::Theories Struct Reference

The Theories class combines all implemented theories and provides a single interface to interact with all theories at once.

`#include <Theories.h>`

Data Structures

- struct ConstantAdder

  Helper functor for addConstants() method.
- struct SimpleSortAdder

  Helper functor for addSimpleSorts() method.

Public Types

- typedef boost::mpl::vector< CoreTheory *, ArithmeticTheory *, BitvectorTheory *, UninterpretedTheory *, BooleanEncodingTheory * >::type Modules

Public Member Functions

- Theories (ParserState *state)
- ~Theories ()
- void addGlobalFormulas (FormulasT &formulas)
- void declareVariable (const std::string &name, const carl::Sort &sort)
- void declareFunction (const std::string &name, const std::vector< carl::Sort > &args, const carl::Sort &sort)
- types::VariableType declareFunctionArgument (const std::pair< std::string, carl::Sort > &arg)
- void defineFunction (const std::string &name, const std::vector< types::VariableType > &arguments, const carl::Sort &sort, const types::TermType &definition)
- types::TermType resolveSymbol (const Identifier &identifier) const
- types::VariableType resolveVariable (const Identifier &identifier) const
- void pushExpressionScope (std::size_t n)
- void popExpressionScope (std::size_t n)
- void pushScriptScope (std::size_t n)
- void popScriptScope (std::size_t n)
- const auto & annotateTerm (const types::TermType &term, const std::vector< Attribute > &attributes)
- void handleLet (const std::string &symbol, const types::TermType &term)
- types::TermType handleITE (const std::vector< types::TermType > &arguments)
- types::TermType handleDistinct (const std::vector< types::TermType > &arguments)
- bool instantiate (const UserFunctionInstantiator &function, const types::VariableType &var, const types::TermType &repl, types::TermType &subject)
- bool instantiateUserFunction (const UserFunctionInstantiator &function, const std::vector< types::TermType > &arguments, types::TermType &result, TheoryError &errors)
- types::TermType functionCall (const Identifier &identifier, const std::vector< types::TermType > &arguments)

Static Public Member Functions

- static void addConstants (qi::symbols< char, types::ConstType > &constants)

  Collects constants from all theory modules.
- static void addSimpleSorts (qi::symbols< char, carl::Sort > &sorts)

  Collects simple sorts from all theory modules.

0.15.516.1 Detailed Description

The Theories class combines all implemented theories and provides a single interface to interact with all theories at once.

0.15.516.2 Member Typedef Documentation

0.15.516.2.1 Modules `typedef boost::mpl::vector< CoreTheory* , ArithmeticTheory* , BitvectorTheory* , UninterpretedTheory* , BooleanEncodingTheory* >::type smtrat::parser::Theories::Modules`

0.15.516.3 Constructor & Destructor Documentation

0.15.516.3.1 Theories() `smtrat::parser::Theories::Theories ( ParserState * state ) [inline]`

0.15.516.3.2 ~Theories() `smtrat::parser::Theories::~Theories ( ) [inline]`

0.15.516.4 Member Function Documentation

0.15.516.4.1 addConstants()

```
static void smtrat::parser::Theories::addConstants (
    qi::symbols< char, types::ConstType > & constants ) [inline], [static]
```

Collects constants from all theory modules.

0.15.516.4.2 addGlobalFormulas()

```
void smtrat::parser::Theories::addGlobalFormulas (
    FormulasT & formulas ) [inline]
```

0.15.516.4.3 addSimpleSorts()

```
static void smtrat::parser::Theories::addSimpleSorts (
    qi::symbols< char, carl::Sort > & sorts ) [inline], [static]
```

Collects simple sorts from all theory modules.

0.15.516.4.4 annotateTerm()

```
const auto& smtrat::parser::Theories::annotateTerm (
    const types::TermType & term,
    const std::vector< Attribute > & attributes ) [inline]
```

0.15.516.4.5 declareFunction()

```
void smtrat::parser::Theories::declareFunction (
    const std::string & name,
    const std::vector< carl::Sort > & args,
    const carl::Sort & sort ) [inline]
```

0.15.516.4.6 declareFunctionArgument()

```
types::VariableType smtrat::parser::Theories::declare↩
FunctionArgument (
    const std::pair< std::string, carl::Sort > & arg ) [inline]
```

0.15.516.4.7 declareVariable()

```
void smtrat::parser::Theories::declareVariable (
    const std::string & name,
    const carl::Sort & sort ) [inline]
```

0.15.516.4.8 defineFunction()

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

Todo check that definition matches the sort

0.15.516.4.9 functionCall()

```
types::TermType smtrat::parser::Theories::functionCall (
    const Identifier & identifier,
    const std::vector< types::TermType > & arguments ) [inline]
```

0.15.516.4.10 handleDistinct()

```
types::TermType smtrat::parser::Theories::handleDistinct (
    const std::vector< types::TermType > & arguments ) [inline]
```

0.15.516.4.11 handleITE()

```
types::TermType smtrat::parser::Theories::handleITE (
const std::vector< types::TermType > & arguments ) [inline]
```

0.15.516.4.12 handleLet()

```
void smtrat::parser::Theories::handleLet (
const std::string & symbol,
const types::TermType & term ) [inline]
```

0.15.516.4.13 instantiate()

```
bool smtrat::parser::Theories::instantiate (
const UserFunctionInstantiator & function,
const types::VariableType & var,
const types::TermType & repl,
types::TermType & subject ) [inline]
```

0.15.516.4.14 instantiateUserFunction()

```
bool smtrat::parser::Theories::instantiateUserFunction (
const UserFunctionInstantiator & function,
const std::vector< types::TermType > & arguments,
types::TermType & result,
TheoryError & errors ) [inline]
```

0.15.516.4.15 popExpressionScope()

```
void smtrat::parser::Theories::popExpressionScope (
std::size_t n ) [inline]
```

0.15.516.4.16 popScriptScope()

```
void smtrat::parser::Theories::popScriptScope (
std::size_t n ) [inline]
```

0.15.516.4.17 pushExpressionScope()

```
void smtrat::parser::Theories::pushExpressionScope (
std::size_t n ) [inline]
```

0.15.516.4.18 pushScriptScope()

```
void smtrat::parser::Theories::pushScriptScope (
std::size_t n ) [inline]
```

0.15.516.4.19 resolveSymbol()

```
types::TermType smtrat::parser::Theories::resolveSymbol (
const Identifier & identifier ) const [inline]
```

0.15.516.4.20 resolveVariable()

```
types::VariableType smtrat::parser::Theories::resolveVariable (
const Identifier & identifier ) const [inline]
```

0.15.517 smtrat::parser::TheoryError Struct Reference

```
#include <Common.h>
```

Public Member Functions

- TheoryError & operator() (const std::string &theory)
- TheoryError & next ()
- template<typename T >
  TheoryError & operator<< (const T &t)

Friends

- std::ostream & operator<< (std::ostream &os, const TheoryError &te)

0.15.517.1 Member Function Documentation

0.15.517.1.1 next()

```
TheoryError& smtrat::parser::TheoryError::next ( ) [inline]
```

0.15.517.1.2 operator()()

```
TheoryError& smtrat::parser::TheoryError::operator() (
    const std::string & theory ) [inline]
```

0.15.517.1.3 operator<<()

```
template<typename T >
TheoryError& smtrat::parser::TheoryError::operator<< (
    const T & t ) [inline]
```

0.15.517.2 Friends And Related Function Documentation

0.15.517.2.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const TheoryError & te ) [friend]
```

0.15.518 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.

0.15.518.1 Field Documentation

0.15.518.1.1 decidedVariables

```
std::vector<Minisat::Var> smtrat::mcsat::TheoryLevel::decided↩
Variables
```

Boolean variables univariate in this theory variable.

0.15.518.1.2 decisionLiteral

```
Minisat::Lit smtrat::mcsat::TheoryLevel::decisionLiteral = Minisat::lit_Undef
```

Literal that assigns this theory variable.

0.15.518.1.3 variable

```
carl::Variable smtrat::mcsat::TheoryLevel::variable = carl::Variable::↩
NO_VARIABLE
```

Theory variable for this level.

0.15.519 smtrat::Module::TheoryPropagation Struct Reference

`#include <Module.h>`

Public Member Functions

- TheoryPropagation (FormulasT &&_premise, const FormulaT &_conclusion)

  Constructor.
- TheoryPropagation ()=delete
- TheoryPropagation (const TheoryPropagation &)=delete
- TheoryPropagation (TheoryPropagation &&_toMove)
- TheoryPropagation & operator= (const TheoryPropagation &_toMove)=delete
- TheoryPropagation & operator= (TheoryPropagation &&_toMove)
- ~TheoryPropagation ()

Data Fields

- FormulasT mPremise

  The constraints under which the propagated constraint holds.
- FormulaT mConclusion

  The propagated constraint.

0.15.519.1 Constructor & Destructor Documentation

0.15.519.1.1 TheoryPropagation() [1/4]

`smtrat::Module::TheoryPropagation::TheoryPropagation ( FormulasT && _premise, const FormulaT & _conclusion ) [inline]`

Constructor.

0.15.519.1.2 TheoryPropagation() [2/4]

`smtrat::Module::TheoryPropagation::TheoryPropagation ( ) [delete]`

0.15.519.1.3 TheoryPropagation() [3/4]

`smtrat::Module::TheoryPropagation::TheoryPropagation ( const TheoryPropagation & ) [delete]`

0.15.519.1.4 TheoryPropagation() [4/4]

`smtrat::Module::TheoryPropagation::TheoryPropagation ( TheoryPropagation && _toMove ) [inline]`

0.15.519.1.5 ~TheoryPropagation()

`smtrat::Module::TheoryPropagation::~TheoryPropagation ( ) [inline]`

0.15.519.2 Member Function Documentation

0.15.519.2.1 operator=() [1/2]

`TheoryPropagation& smtrat::Module::TheoryPropagation::operator= ( const TheoryPropagation & _toMove ) [delete]`

0.15.519.2.2 operator=() [2/2]

`TheoryPropagation& smtrat::Module::TheoryPropagation::operator= ( TheoryPropagation && _toMove ) [inline]`

0.15.519.3 Field Documentation

0.15.519.3.1 mConclusion `FormulaT smtrat::Module::TheoryPropagation::mConclusion`

The propagated constraint.

0.15.519.3.2 mPremise `FormulasT smtrat::Module::TheoryPropagation::mPremise`

The constraints under which the propagated constraint holds.

0.15.520 smtrat::TheoryVarSchedulerStatic< vot > Class Template Reference

Schedules theory variables statically.

`#include <VarSchedulerMcsat.h>`

Public Member Functions

- template<typename BaseModule >
  TheoryVarSchedulerStatic (BaseModule &baseModule)
- void insert (Minisat::Var var)
- bool empty ()
- Minisat::Lit pop ()
- void print () const
- template<typename Constraints >
  void rebuildTheoryVars (const Constraints &c)
- size_t level ()

  Level of the next theory variable.
- size_t univariateLevel (Minisat::Var v)

  Returns the level in which the given constraint is univariate.
- void rebuild ()

  Rebuild heap.
- void increaseActivity (Minisat::Var)
- void decreaseActivity (Minisat::Var)
- void rebuildActivities ()
- void attachClause (Minisat::CRef)
- void detachClause (Minisat::CRef)
- void relocateClauses (std::function< void(Minisat::CRef &)>)

Protected Attributes

- std::function< bool(Minisat::Var)> isTheoryVar
- std::function< carl::Variable(Minisat::Var)> carlVar
- std::function< Minisat::Var(carl::Variable)> minisatVar
- std::function< Model()> currentModel
- std::function< double(Minisat::Var)> getActivity
- std::function< char(Minisat::Var)> getPolarity
- std::function< void(Minisat::Var, bool)> setPolarity
- std::function< bool(Minisat::Var)> isDecisionVar
- std::function< bool(Minisat::Var)> isBoolValueUndef
- std::function< bool(Minisat::Var)> isTheoryAbstraction
- std::function< const FormulaT &(Minisat::Var)> reabstractVariable
- std::function< const FormulaT &(Minisat::Lit)> reabstractLiteral
- std::function< const Minisat::Clause &(Minisat::CRef)> getClause
- std::function< Minisat::lbool(Minisat::Var)> getBoolVarValue
- std::function< Minisat::lbool(Minisat::Lit)> getBoolLitValue

- std::function< unsigned(const FormulaT &)> currentlySatisfiedByBackend
- std::function< Minisat::Var(const FormulaT &)> abstractVariable
- std::function< const Minisat::Lit(const FormulaT &)> abstractLiteral
- std::function< bool(const FormulaT &)> isAbstractedFormula

0.15.520.1 Detailed Description

template<mcsat::VariableOrdering vot>
class smtrat::TheoryVarSchedulerStatic< vot >

Schedules theory variables statically.
Should not be used directly.

0.15.520.2 Constructor & Destructor Documentation

0.15.520.2.1 TheoryVarSchedulerStatic()

```
template<mcsat::VariableOrdering vot>
template<typename BaseModule >
smtrat::TheoryVarSchedulerStatic< vot >::TheoryVarSchedulerStatic (
            BaseModule & baseModule )  [inline], [explicit]
```

0.15.520.3 Member Function Documentation

0.15.520.3.1 attachClause()

```
void smtrat::VarSchedulerBase::attachClause (
            Minisat::CRef  )  [inline], [inherited]
```

0.15.520.3.2 decreaseActivity()

```
void smtrat::VarSchedulerBase::decreaseActivity (
            Minisat::Var  )  [inline], [inherited]
```

0.15.520.3.3 detachClause()

```
void smtrat::VarSchedulerBase::detachClause (
            Minisat::CRef  )  [inline], [inherited]
```

0.15.520.3.4 empty()

```
template<mcsat::VariableOrdering vot>
bool smtrat::TheoryVarSchedulerStatic< vot >::empty ( )  [inline]
```

0.15.520.3.5 increaseActivity()

```
void smtrat::VarSchedulerBase::increaseActivity (
            Minisat::Var  )  [inline], [inherited]
```

0.15.520.3.6 insert()

```
template<mcsat::VariableOrdering vot>
void smtrat::TheoryVarSchedulerStatic< vot >::insert (
            Minisat::Var var )  [inline]
```

0.15.520.3.7 level()

```
template<mcsat::VariableOrdering vot>
size_t smtrat::TheoryVarSchedulerStatic< vot >::level ( )  [inline]
```

Level of the next theory variable.

0.15.520.3.8 pop()

```
template<mcsat::VariableOrdering vot>
Minisat::Lit smtrat::TheoryVarSchedulerStatic< vot >::pop ( ) [inline]
```

0.15.520.3.9 print()

```
template<mcsat::VariableOrdering vot>
void smtrat::TheoryVarSchedulerStatic< vot >::print ( ) const [inline]
```

0.15.520.3.10 rebuild()

```
void smtrat::VarSchedulerBase::rebuild ( ) [inline], [inherited]
```

Rebuild heap.

0.15.520.3.11 rebuildActivities()

```
void smtrat::VarSchedulerBase::rebuildActivities ( ) [inline],
[inherited]
```

0.15.520.3.12 rebuildTheoryVars()

```
template<mcsat::VariableOrdering vot>
template<typename Constraints >
void smtrat::TheoryVarSchedulerStatic< vot >::rebuildTheoryVars (
            const Constraints & c ) [inline]
```

0.15.520.3.13 relocateClauses()

```
void smtrat::VarSchedulerBase::relocateClauses (
            std::function< void(Minisat::CRef &)> ) [inline], [inherited]
```

0.15.520.3.14 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.

0.15.520.4 Field Documentation

0.15.520.4.1 abstractLiteral

```
std::function<const Minisat::Lit(const FormulaT&)> smtrat::Var↩
SchedulerBase::abstractLiteral [protected], [inherited]
```

0.15.520.4.2 abstractVariable

```
std::function< Minisat::Var(const FormulaT&)> smtrat::VarScheduler↩
Base::abstractVariable [protected], [inherited]
```

0.15.520.4.3 carlVar

```
std::function<carl::Variable(Minisat::Var)> smtrat::VarSchedulerMcsat↩
Base::carlVar [protected], [inherited]
```

0.15.520.4.4 currentlySatisfiedByBackend

```
std::function<unsigned(const FormulaT&)> smtrat::Var↩
SchedulerBase::currentlySatisfiedByBackend [protected], [inherited]
```

0.15.520.4.5 currentModel

```
std::function<Model()> smtrat::VarSchedulerMcsatBase::currentModel
[protected], [inherited]
```

0.15.520.4.6 getActivity

std::function<double(Minisat::Var)> smtrat::VarSchedulerBase::getActivity [protected], [inherited]

0.15.520.4.7 getBoolLitValue

std::function<Minisat::lbool(Minisat::Lit)> smtrat::VarSchedulerBase::getBoolLitValue [protected], [inherited]

0.15.520.4.8 getBoolVarValue

std::function<Minisat::lbool(Minisat::Var)> smtrat::VarSchedulerBase::getBoolVarValue [protected], [inherited]

0.15.520.4.9 getClause

std::function<const Minisat::Clause&(Minisat::CRef)> smtrat::VarSchedulerBase::getClause [protected], [inherited]

0.15.520.4.10 getPolarity

std::function<char(Minisat::Var)> smtrat::VarSchedulerBase::getPolarity [protected], [inherited]

0.15.520.4.11 isAbstractedFormula

std::function<bool(const FormulaT&)> smtrat::VarSchedulerBase::isAbstractedFormula [protected], [inherited]

0.15.520.4.12 isBoolValueUndef

std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isBoolValueUndef [protected], [inherited]

0.15.520.4.13 isDecisionVar

std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isDecisionVar [protected], [inherited]

0.15.520.4.14 isTheoryAbstraction

std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isTheoryAbstraction [protected], [inherited]

0.15.520.4.15 isTheoryVar

std::function<bool(Minisat::Var)> smtrat::VarSchedulerMcsatBase::isTheoryVar [protected], [inherited]

0.15.520.4.16 minisatVar

std::function<Minisat::Var(carl::Variable)> smtrat::VarSchedulerMcsatBase::minisatVar [protected], [inherited]

0.15.520.4.17 reabstractLiteral

std::function<const FormulaT&(Minisat::Lit)> smtrat::VarSchedulerBase::reabstractLiteral [protected], [inherited]

0.15.520.4.18 reabstractVariable

std::function<const FormulaT&(Minisat::Var)> smtrat::VarSchedulerBase::reabstractVariable [protected], [inherited]

0.15.520.4.19 setPolarity

std::function<void(Minisat::Var,bool)> smtrat::VarSchedulerBase::setPolarity [protected], [inherited]

0.15.521 smtrat::ThreadPool Class Reference

`#include <ThreadPool.h>`

Public Member Functions

- ThreadPool (std::size_t maxThreads)
- ~ThreadPool ()
- Answer runBackends (const std::vector< Module * > &_modules, bool _final, bool _full, carl::Variable _objective)

0.15.521.1 Constructor & Destructor Documentation

0.15.521.1.1 ThreadPool()

```
smtrat::ThreadPool::ThreadPool (
    std::size_t maxThreads ) [inline]
```

0.15.521.1.2 ~ThreadPool()

```
smtrat::ThreadPool::~ThreadPool ( ) [inline]
```

0.15.521.2 Member Function Documentation

0.15.521.2.1 runBackends()

```
Answer smtrat::ThreadPool::runBackends (
    const std::vector< Module * > & _modules,
    bool _final,
    bool _full,
    carl::Variable _objective )
```

Parameters

_modules	
_final	
_full	
_objective	

0.15.522 smtrat::cad::debug::TikzBasePrinter Class Reference

`#include <TikzHistoryPrinter.h>`

Public Member Functions

- virtual void addNode (std::size_t level, const std::string &parent, const std::string &node, const std::string &data)=0
- virtual void addEdge (const std::string &src, const std::string &dst, const std::string &data)=0
- void step ()
- virtual void layout ()=0
- virtual void writeTo (std::ostream &os, int xBase) const =0
- virtual std::size_t width () const =0

Protected Member Functions

- std::string printableID (const std::string &raw, const std::string &prefix, std::map< std::string, std::string > &map) const

Protected Attributes

- std::size_t mStep

0.15.522.1 Member Function Documentation

0.15.522.1.1 addEdge()

```
virtual void smtrat::cad::debug::TikzBasePrinter::addEdge (
    const std::string & src,
    const std::string & dst,
    const std::string & data ) [pure virtual]
```

Implemented in smtrat::cad::debug::TikzTreePrinter, and smtrat::cad::debug::TikzDAGPrinter.

0.15.522.1.2 addNode()

```
virtual void smtrat::cad::debug::TikzBasePrinter::addNode (
    std::size_t level,
    const std::string & parent,
    const std::string & node,
    const std::string & data ) [pure virtual]
```

Implemented in smtrat::cad::debug::TikzTreePrinter, and smtrat::cad::debug::TikzDAGPrinter.

0.15.522.1.3 layout()

```
virtual void smtrat::cad::debug::TikzBasePrinter::layout ( ) [pure virtual]
```

Implemented in smtrat::cad::debug::TikzTreePrinter, and smtrat::cad::debug::TikzDAGPrinter.

0.15.522.1.4 printableID()

```
std::string smtrat::cad::debug::TikzBasePrinter::printableID (
    const std::string & raw,
    const std::string & prefix,
    std::map< std::string, std::string > & map ) const [inline], [protected]
```

0.15.522.1.5 step()

```
void smtrat::cad::debug::TikzBasePrinter::step ( ) [inline]
```

0.15.522.1.6 width()

```
virtual std::size_t smtrat::cad::debug::TikzBasePrinter::width ( ) const
[pure virtual]
```

Implemented in smtrat::cad::debug::TikzTreePrinter, and smtrat::cad::debug::TikzDAGPrinter.

0.15.522.1.7 writeTo()

```
virtual void smtrat::cad::debug::TikzBasePrinter::writeTo (
    std::ostream & os,
    int xBase ) const [pure virtual]
```

Implemented in smtrat::cad::debug::TikzTreePrinter, and smtrat::cad::debug::TikzDAGPrinter.

0.15.522.2 Field Documentation

0.15.522.2.1 mStep

```
std::size_t smtrat::cad::debug::TikzBasePrinter::mStep [protected]
```

0.15.523 smtrat::cad::debug::TikzDAGPrinter Class Reference

```
#include <TikzHistoryPrinter.h>
```

Public Member Functions

- void addNode (std::size_t level, const std::string &, const std::string &node, const std::string &data) override
- void addEdge (const std::string &src, const std::string &dst, const std::string &data) override
- void layout () override
- void writeTo (std::ostream &os, int xBase) const override
- std::size_t width () const override
- void step ()

Protected Member Functions

- std::string printableID (const std::string &raw, const std::string &prefix, std::map< std::string, std::string > &map) const

Protected Attributes

- std::size_t mStep

0.15.523.1 Member Function Documentation

0.15.523.1.1 addEdge()

```
void smtrat::cad::debug::TikzDAGPrinter::addEdge (
    const std::string & src,
    const std::string & dst,
    const std::string & data ) [inline], [override], [virtual]
```

Implements smtrat::cad::debug::TikzBasePrinter.

0.15.523.1.2 addNode()

```
void smtrat::cad::debug::TikzDAGPrinter::addNode (
    std::size_t level,
    const std::string & ,
    const std::string & node,
    const std::string & data ) [inline], [override], [virtual]
```

Implements smtrat::cad::debug::TikzBasePrinter.

0.15.523.1.3 layout()

```
void smtrat::cad::debug::TikzDAGPrinter::layout ( ) [inline], [override], [virtual]
```

Implements smtrat::cad::debug::TikzBasePrinter.

0.15.523.1.4 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], [inherited]
```

0.15.523.1.5 step()

```
void smtrat::cad::debug::TikzBasePrinter::step ( ) [inline], [inherited]
```

0.15.523.1.6 width()

```
std::size_t smtrat::cad::debug::TikzDAGPrinter::width ( ) const [inline], [override], [virtual]
```

Implements smtrat::cad::debug::TikzBasePrinter.

0.15.523.1.7 writeTo()

```
void smtrat::cad::debug::TikzDAGPrinter::writeTo (
            std::ostream & os,
            int xBase ) const [inline], [override], [virtual]
```

Implements smtrat::cad::debug::TikzBasePrinter.

0.15.523.2 Field Documentation

0.15.523.2.1 mStep

```
std::size_t smtrat::cad::debug::TikzBasePrinter::mStep [protected], [inherited]
```

0.15.524 smtrat::cad::debug::TikzHistoryPrinter Class Reference

```
#include <TikzHistoryPrinter.h>
```

Public Member Functions

- template<typename Printer >
  void configure (const std::string &name)
- template<typename ID1 , typename ID2 , typename T >
  void addNode (const std::string &printer, std::size_t level, const ID1 &parent, const ID2 &node, const T &data)
- template<typename ID1 , typename ID2 , typename T >
  void addEdge (const std::string &printer, const ID1 &src, const ID2 &dst, const T &data)
- template<typename L >
  void addLifting (const L &l)
- template<typename P >
  void addProjection (const P &p)
- void step ()
- void layout ()
- void writeTo (const std::string &filename) const

0.15.524.1 Member Function Documentation

0.15.524.1.1 addEdge()

```
template<typename ID1 , typename ID2 , typename T >
void smtrat::cad::debug::TikzHistoryPrinter::addEdge (
            const std::string & printer,
            const ID1 & src,
            const ID2 & dst,
            const T & data ) [inline]
```

0.15.524.1.2 addLifting()

```
template<typename L >
void smtrat::cad::debug::TikzHistoryPrinter::addLifting (
            const L & l ) [inline]
```

0.15.524.1.3 addNode()

```
template<typename ID1 , typename ID2 , typename T >
void smtrat::cad::debug::TikzHistoryPrinter::addNode (
            const std::string & printer,
            std::size_t level,
            const ID1 & parent,
            const ID2 & node,
            const T & data ) [inline]
```

0.15.524.1.4 addProjection()

```
template<typename P >
void smtrat::cad::debug::TikzHistoryPrinter::addProjection (
            const P & p )  [inline]
```

0.15.524.1.5 configure()

```
template<typename Printer >
void smtrat::cad::debug::TikzHistoryPrinter::configure (
            const std::string & name )  [inline]
```

0.15.524.1.6 layout()

```
void smtrat::cad::debug::TikzHistoryPrinter::layout ( )  [inline]
```

0.15.524.1.7 step()

```
void smtrat::cad::debug::TikzHistoryPrinter::step ( )  [inline]
```

0.15.524.1.8 writeTo()

```
void smtrat::cad::debug::TikzHistoryPrinter::writeTo (
            const std::string & filename ) const  [inline]
```

0.15.525 smtrat::cad::debug::TikzTreePrinter Class Reference

```
#include <TikzHistoryPrinter.h>
```

Public Member Functions

- void setRoot (const std::string &node, const std::string &data)
- void addNode (std::size_t level, const std::string &parent, const std::string &node, const std::string &data) override
- void addEdge (const std::string &src, const std::string &dst, const std::string &data) override
- void layout () override
- void writeTo (std::ostream &os, int xBase) const override
- virtual std::size_t width () const override
- void step ()

Protected Member Functions

- std::string printableID (const std::string &raw, const std::string &prefix, std::map< std::string, std::string > &map) const

Protected Attributes

- std::size_t mStep

0.15.525.1 Member Function Documentation

0.15.525.1.1 addEdge()

```
void smtrat::cad::debug::TikzTreePrinter::addEdge (
            const std::string & src,
            const std::string & dst,
            const std::string & data )  [inline], [override], [virtual]
```

Implements smtrat::cad::debug::TikzBasePrinter.

0.15.525.1.2 addNode()

```
void smtrat::cad::debug::TikzTreePrinter::addNode (
    std::size_t level,
    const std::string & parent,
    const std::string & node,
    const std::string & data ) [inline], [override], [virtual]
```

Implements smtrat::cad::debug::TikzBasePrinter.

0.15.525.1.3 layout()

```
void smtrat::cad::debug::TikzTreePrinter::layout ( ) [inline], [override], [virtual]
```

Implements smtrat::cad::debug::TikzBasePrinter.

0.15.525.1.4 printableID()

```
std::string smtrat::cad::debug::TikzBasePrinter::printableID (
    const std::string & raw,
    const std::string & prefix,
    std::map< std::string, std::string > & map ) const [inline], [protected], [inherited]
```

0.15.525.1.5 setRoot()

```
void smtrat::cad::debug::TikzTreePrinter::setRoot (
    const std::string & node,
    const std::string & data ) [inline]
```

0.15.525.1.6 step()

```
void smtrat::cad::debug::TikzBasePrinter::step ( ) [inline], [inherited]
```

0.15.525.1.7 width()

```
virtual std::size_t smtrat::cad::debug::TikzTreePrinter::width ( ) const [inline], [override], [virtual]
```

Implements smtrat::cad::debug::TikzBasePrinter.

0.15.525.1.8 writeTo()

```
void smtrat::cad::debug::TikzTreePrinter::writeTo (
    std::ostream & os,
    int xBase ) const [inline], [override], [virtual]
```

Implements smtrat::cad::debug::TikzBasePrinter.

0.15.525.2 Field Documentation

0.15.525.2.1 mStep

```
std::size_t smtrat::cad::debug::TikzBasePrinter::mStep [protected], [inherited]
```

0.15.526 benchmax::Tool Class Reference

Base class for any tool.

`#include <Tool.h>`

Public Member Functions

- Tool ()=delete
- Tool (const Tool &)=delete
- Tool (Tool &&)=delete
- Tool (const fs::path &binary, const std::string &arguments)

  Creates a generic tool from a binary and command line arguments.
- Tool (const std::string &name, const fs::path &binary, const std::string &arguments)

Creates a named tool from a binary and command line arguments.

- virtual ~Tool ()=default
- Tool & operator= (const Tool &)=delete
- Tool & operator= (Tool &&)=delete
- std::string name () const

  Common name of this tool.
- fs::path binary () const

  Full path to the binary.
- const std::map< std::string, std::string > & attributes () const

  A set of attributes, for example compilation options.
- std::vector< std::string > resolveDependencies () const

  Get dependencies of binary required to run it (via ldd)
- std::size_t attributeHash () const

  Hash of the attributes.
- virtual std::string getCommandline (const std::string &file) const

  Compose commandline for this tool and the given input file.
- virtual std::string getCommandline (const std::string &file, const std::string &localBinary) const

  Compose commandline for this tool with another binary name and the given input file.
- virtual std::optional< std::string > parseCommandline (const std::string &cmdline) const

  Compose commandline for this tool and the given input file.
- virtual bool canHandle (const fs::path &) const

  Checks whether this cool can handle this file type.
- virtual void additionalResults (const fs::path &, BenchmarkResult &) const

  Recover additional results from the tool output.

Protected Attributes

- std::string mName

  (Non-unique) identifier for the tool.
- fs::path mBinary

  Path to the binary.
- std::string mArguments

  Command line arguments that should be passed to the binary.
- std::map< std::string, std::string > mAttributes

  Attributes of the tool obtained by introspection of the binary.

Friends

- bool operator< (const Tool &lhs, const Tool &rhs)

  Compare two tools.

0.15.526.1 Detailed Description

Base class for any tool.
A tool represents some executable that can be run with some input file. A tool is responsible for

- deciding it is applicable for a given file extension,
- building a command line to execute it and
- parse additional results from stdout / stderr after it has run.

A tool is not to be copied or moved around but should only exist a single time.

0.15.526.2 Constructor & Destructor Documentation

0.15.526.2.1 Tool() [1/5] benchmax::Tool::Tool ( ) [delete]

0.15.526.2.2 Tool() [2/5] benchmax::Tool::Tool (const Tool &) [delete]

0.15.526.2.3 Tool() [3/5] benchmax::Tool::Tool (Tool &&) [delete]

0.15.526.2.4 Tool() [4/5] benchmax::Tool::Tool (const fs::path & *binary*, const std::string & *arguments*) [inline]

Creates a generic tool from a binary and command line arguments.

0.15.526.2.5 Tool() [5/5] benchmax::Tool::Tool (const std::string & *name*, const fs::path & *binary*, const std::string & *arguments*) [inline]

Creates a named tool from a binary and command line arguments.

0.15.526.2.6 ~Tool() virtual benchmax::Tool::~Tool ( ) [virtual], [default]

0.15.526.3 Member Function Documentation

0.15.526.3.1 additionalResults() virtual void benchmax::Tool::additionalResults (const fs::path & , BenchmarkResult &) const [inline], [virtual]

Recover additional results from the tool output.

Reimplemented in benchmax::Z3, benchmax::SMTRAT_Analyzer, benchmax::SMTRAT, benchmax::Minisatp, benchmax::Minisat, and benchmax::MathSAT.

0.15.526.3.2 attributeHash() std::size_t benchmax::Tool::attributeHash ( ) const [inline]

Hash of the attributes.

0.15.526.3.3 attributes() const std::map<std::string,std::string>& benchmax::Tool::attributes ( ) const [inline]

A set of attributes, for example compilation options.

0.15.526.3.4 binary() fs::path benchmax::Tool::binary ( ) const [inline]

Full path to the binary.

0.15.526.3.5 canHandle()

```
virtual bool benchmax::Tool::canHandle (
    const fs::path & ) const [inline], [virtual]
```

Checks whether this cool can handle this file type.

Reimplemented in benchmax::Z3, benchmax::SMTRAT_OPB, benchmax::SMTRAT_Analyzer, benchmax::SMTRAT, benchmax::Minisatp, benchmax::Minisat, and benchmax::MathSAT.

0.15.526.3.6 getCommandline() [1/2]

```
virtual std::string benchmax::Tool::getCommandline (
    const std::string & file ) const [inline], [virtual]
```

Compose commandline for this tool and the given input file.

0.15.526.3.7 getCommandline() [2/2]

```
virtual std::string benchmax::Tool::getCommandline (
    const std::string & file,
    const std::string & localBinary ) const [inline], [virtual]
```

Compose commandline for this tool with another binary name and the given input file.

0.15.526.3.8 name()

```
std::string benchmax::Tool::name ( ) const [inline]
```

Common name of this tool.

0.15.526.3.9 operator=() [1/2]

```
Tool& benchmax::Tool::operator= (
    const Tool & ) [delete]
```

0.15.526.3.10 operator=() [2/2]

```
Tool& benchmax::Tool::operator= (
    Tool && ) [delete]
```

0.15.526.3.11 parseCommandline()

```
virtual std::optional<std::string> benchmax::Tool::parse↩
Commandline (
    const std::string & cmdline ) const [inline], [virtual]
```

Compose commandline for this tool and the given input file.

0.15.526.3.12 resolveDependencies()

```
std::vector<std::string> benchmax::Tool::resolveDependencies
( ) const [inline]
```

Get dependencies of binary required to run it (via ldd)

0.15.526.4 Friends And Related Function Documentation

0.15.526.4.1 operator<

```
bool operator< (
    const Tool & lhs,
    const Tool & rhs ) [friend]
```

Compare two tools.

0.15.526.5 Field Documentation

0.15.526.5.1 mArguments

```
std::string benchmax::Tool::mArguments [protected]
```

Command line arguments that should be passed to the binary.

0.15.526.5.2 mAttributes `std::map<std::string,std::string> benchmax::Tool::mAttributes [protected]`

Attributes of the tool obtained by introspection of the binary.

0.15.526.5.3 mBinary `fs::path benchmax::Tool::mBinary [protected]`

Path to the binary.

0.15.526.5.4 mName `std::string benchmax::Tool::mName [protected]`

(Non-unique) identifier for the tool.

0.15.527 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.

0.15.527.1 Detailed Description

Tool-related settings.

0.15.527.2 Field Documentation

0.15.527.2.1 collect_statistics `bool benchmax::settings::ToolSettings::collect_statistics`

Whether or not to collect statistics.

0.15.527.2.2 tools_common_prefix `std::filesystem::path benchmax::settings::ToolSettings↩ ::tools_common_prefix`

Common prefix of tool binaries to simplify output.

0.15.527.2.3 tools_generic `std::vector<std::filesystem::path> benchmax::settings::ToolSettings↩ ::tools_generic`

Generic tools.

0.15.527.2.4 tools_mathsat `std::vector<std::filesystem::path> benchmax::settings::ToolSettings↩ ::tools_mathsat`

MathSAT with SMT-LIB interface.

0.15.527.2.5 tools_minisat `std::vector<std::filesystem::path> benchmax::settings::ToolSettings↩ ::tools_minisat`

Minisat with DIMACS interface.

0.15.527.2.6 tools_minisatp `std::vector<std::filesystem::path> benchmax::settings::Tool↩ Settings::tools_minisatp`

Minisatp with OPB interface.

0.15.527.2.7 tools_smtrat `std::vector<std::filesystem::path> benchmax::settings::ToolSettings↩ ::tools_smtrat`

SMT-RAT with SMT-LIB interface.

0.15.527.2.8 tools_smtrat_analyzer `std::vector<std::filesystem::path> benchmax::settings::↩ ToolSettings::tools_smtrat_analyzer`

SMT-RAT formula analyzer.

0.15.527.2.9 tools_smtrat_opb `std::vector<std::filesystem::path> benchmax::settings::Tool↩ Settings::tools_smtrat_opb`

SMT-RAT with OPB interface.

0.15.527.2.10 tools_z3 `std::vector<std::filesystem::path> benchmax::settings::ToolSettings↩ ::tools_z3`

z3 with SMT-LIB interface.

0.15.528 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 ()
- std::optional< FormulaT > encode (const ConstraintT &constraint)

  Encodes an arbitrary constraint.

Data Fields

- std::size_t problem_size

Protected Member Functions

- std::optional< FormulaT > doEncode (const ConstraintT &constraint)
- FormulaT generateVarChain (const std::set< carl::Variable > &vars, carl::FormulaType type)

0.15.528.1 Detailed Description

TotalizerEncoder implements the Totalizer encoding as described in "Incremental Cardinality Constraints for Max↩ SAT" by Martins et al https://doi.org/10.1007/978-3-319-10428-7_39.

0.15.528.2 Constructor & Destructor Documentation

0.15.528.2.1 TotalizerEncoder() `smtrat::TotalizerEncoder::TotalizerEncoder ( ) [inline]`

0.15.528.2.2 ~TotalizerEncoder() `smtrat::TotalizerEncoder::~TotalizerEncoder ( )`

0.15.528.3 Member Function Documentation

0.15.528.3.1 canEncode()

```
bool smtrat::TotalizerEncoder::canEncode (
    const ConstraintT & constraint ) [virtual]
```

Implements smtrat::PseudoBoolEncoder.

0.15.528.3.2 doEncode()

```
std::optional< FormulaT > smtrat::TotalizerEncoder::doEncode (
    const ConstraintT & constraint ) [protected], [virtual]
```

Implements smtrat::PseudoBoolEncoder.

0.15.528.3.3 encode()

```
std::optional< FormulaT > smtrat::PseudoBoolEncoder::encode (
    const ConstraintT & constraint ) [inherited]
```

Encodes an arbitrary constraint.

Returns

encoded formula

0.15.528.3.4 encodingSize()

```
Rational smtrat::TotalizerEncoder::encodingSize (
    const ConstraintT & constraint ) [virtual]
```

Reimplemented from smtrat::PseudoBoolEncoder.

0.15.528.3.5 generateVarChain()

```
FormulaT smtrat::PseudoBoolEncoder::generateVarChain (
    const std::set< carl::Variable > & vars,
    carl::FormulaType type ) [protected], [inherited]
```

0.15.528.3.6 name()

```
std::string smtrat::TotalizerEncoder::name ( ) [inline], [virtual]
```

Reimplemented from smtrat::PseudoBoolEncoder.

0.15.528.4 Field Documentation

0.15.528.4.1 problem_size `std::size_t smtrat::PseudoBoolEncoder::problem_size [inherited]`

0.15.529 smtrat::TotalizerTree Class Reference

`#include <TotalizerEncoder.h>`

Public Member Functions

- TotalizerTree (const std::set< carl::Variable > &variables)
- ~TotalizerTree ()
- bool isLeaf ()
- TotalizerTree & left ()
- TotalizerTree & right ()
- std::vector< carl::Variable > variables ()

0.15.529.1 Constructor & Destructor Documentation

0.15.529.1.1 TotalizerTree() `smtrat::TotalizerTree::TotalizerTree ( const std::set< carl::Variable > & variables )`

0.15.529.1.2 ~TotalizerTree() `smtrat::TotalizerTree::~TotalizerTree ( ) [inline]`

0.15.529.2 Member Function Documentation

0.15.529.2.1 isLeaf() `bool smtrat::TotalizerTree::isLeaf ( ) [inline]`

0.15.529.2.2 left() `TotalizerTree& smtrat::TotalizerTree::left ( ) [inline]`

0.15.529.2.3 right() `TotalizerTree& smtrat::TotalizerTree::right ( ) [inline]`

0.15.529.2.4 variables() `std::vector<carl::Variable> smtrat::TotalizerTree::variables ( ) [inline]`

0.15.530 smtrat::covering_ng::formula::node_ds::TRUE Struct Reference

`#include <FormulaEvaluation.h>`

0.15.531 smtrat::cad::projection_compare::type Struct Reference

`#include <ProjectionComparator.h>`

0.15.532 smtrat::cad::sample_compare::type Struct Reference

`#include <SampleComparator.h>`

0.15.533 smtrat::UFCegarModule< Settings > Class Template Reference

`#include <UFCegarModule.h>`

Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- UFCegarModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=nullptr)
- ~UFCegarModule ()
- bool addCore (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- void removeCore (ModuleInput::const_iterator _subformula)

  Removes the subformula of the received formula at the given position to the considered ones of this module.
- void updateModel () const

  Updates the current assignment into the model.
- Answer checkCore ()

  Checks the received formula for consistency.
- bool inform (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void 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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.
- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)
- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.

- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

  Copies the infeasible subsets of the passed formula.
- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

Gets all InformationRelevantFormulas.

- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.533.1 Member Typedef Documentation

0.15.533.1.1 SettingsType

```
template<typename Settings >
typedef Settings smtrat::UFCegarModule< Settings >::SettingsType
```

0.15.533.2 Member Enumeration Documentation

0.15.533.2.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

0.15.533.3 Constructor & Destructor Documentation

0.15.533.3.1 UFCegarModule()

```
template<typename Settings >
smtrat::UFCegarModule< Settings >::UFCegarModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

0.15.533.3.2 ~UFCegarModule()

```
template<typename Settings >
smtrat::UFCegarModule< Settings >::~UFCegarModule ( )
```

0.15.533.4 Member Function Documentation

0.15.533.4.1 add()

```
bool smtrat::Module::add (
            ModuleInput::const_iterator _subformula ) [inherited]
```

The module has to take the given sub-formula of the received formula into account.

Parameters

_subformula	The sub-formula to take additionally into account.

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

0.15.533.4.2 addConstraintToInform()

```
void smtrat::Module::addConstraintToInform (
            const FormulaT & _constraint ) [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

0.15.533.4.3 addCore() [1/2]

```
virtual bool smtrat::Module::addCore (
    [[maybe_unused] ] 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.

0.15.533.4.4 addCore() [2/2]

```
template<typename Settings >
bool smtrat::UFCegarModule< Settings >::addCore (
    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.

0.15.533.4.5 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula
(
    const FormulaT & formula ) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.533.4.6 addLemma()

```
void smtrat::Module::addLemma (
    const FormulaT & _lemma,
    const LemmaType & _lt = LemmaType::NORMAL,
    const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline],
[inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.533.4.7 addOrigin()

```
void smtrat::Module::addOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.533.4.8 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.533.4.9 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.533.4.10 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.533.4.11 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.

0.15.533.4.12 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns

All satisfying assignments, if existent.

0.15.533.4.13 anAnswerFound()

```
bool smtrat::Module::anAnswerFound ( ) const [inline], [protected],
[inherited]
```

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.533.4.14 answerFound()

```
const smtrat::Conditionals& smtrat::Module::answerFound ( ) const
[inline], [inherited]
```

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.533.4.15 backendsModel()

```
const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]
```

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.533.4.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.533.4.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.533.4.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.533.4.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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.

Parameters

_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

0.15.533.4.20 check()

```
Answer smtrat::Module::check (
    bool _final = false,
    bool _full = true,
    carl::Variable _objective = carl::Variable::NO_VARIABLE ) [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.533.4.21 checkCore()

```
template<typename Settings >
Answer smtrat::UFCegarModule< Settings >::checkCore ( ) [virtual]
```

Checks the received formula for consistency.

Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.533.4.22 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.

0.15.533.4.23 checkModel() `unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]`

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.533.4.24 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.533.4.25 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

0.15.533.4.26 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

0.15.533.4.27 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

0.15.533.4.28 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.533.4.29 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

0.15.533.4.30 collectOrigins() [2/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.
If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.533.4.31 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

0.15.533.4.32 constraintsToInform()

`const carl::FastSet<FormulaT>& smtrat::Module::constraintsToInform ( ) const [inline], [inherited]`

Returns

The constraints which the backends must be informed about.

0.15.533.4.33 currentlySatisfied()

`virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.533.4.34 currentlySatisfiedByBackend()

`unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.533.4.35 deinform()

`void smtrat::Module::deinform ( const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.533.4.36 deinformCore()

`virtual void smtrat::Module::deinformCore ( [[maybe_unused] ] 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.

0.15.533.4.37 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

0.15.533.4.38 eraseSubformulaFromPassedFormula()

```
ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (
            ModuleInput::iterator _subformula,
            bool _ignoreOrigins = false ) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_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 >.

0.15.533.4.39 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

0.15.533.4.40 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
            const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

0.15.533.4.41 firstSubformulaToPass()

```
ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]
```

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.533.4.42 firstUncheckedReceivedSubformula() `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.533.4.43 freeSplittingVariable() `static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.533.4.44 generateTrivialInfeasibleSubset() `void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.533.4.45 getBackendsAllModels() `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.533.4.46 getBackendsModel() `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.533.4.47 getInfeasibleSubsets() [1/2] `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.533.4.48 getInfeasibleSubsets() [2/2] `std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets ( const Module & _backend ) const [protected], [inherited]`

Get the infeasible subsets the given backend provides.
Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

Parameters

_backend	The backend from which to obtain the infeasible subsets.

Returns

The infeasible subsets the given backend provides.

0.15.533.4.49 getInformationRelevantFormulas() `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.533.4.50 getModelEqualities() `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.533.4.51 getOrigins() [1/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.533.4.52 getOrigins() [2/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.533.4.53 getOrigins() [3/3] `const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]`

Gets the origins of the passed formula at the given position.

Parameters

_formula	The position of a formula in the passed formulas.

Returns

The origins of the passed formula at the given position.

0.15.533.4.54 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.

0.15.533.4.55 hasLemmas() `bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

0.15.533.4.56 hasValidInfeasibleSubset() `bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]`

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.533.4.57 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.533.4.58 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasible↩ Subsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.533.4.59 inform() `bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.
It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

Parameters

_constraint	The constraint to inform about.

Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

0.15.533.4.60 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.

0.15.533.4.61 informCore() `virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.533.4.62 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.

0.15.533.4.63 init() `void smtrat::Module::init ( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not be called twice and only before the first runBackends call.
Reimplemented in smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntBlastModule< Settings >, smtrat::FPPModule< Settings >, smtrat::CurryModule< Settings >, and smtrat::BVModule< Settings >.

0.15.533.4.64 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.533.4.65 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.

0.15.533.4.66 isPreprocessor() `bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.533.4.67 lemmas() `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.533.4.68 merge()

```
std::vector< FormulaT > smtrat::Module::merge (
    const std::vector< FormulaT > & _vecSetA,
    const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]
```

Merges the two vectors of sets into the first one.
({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

Parameters

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns

The vector being the two given vectors merged.

0.15.533.4.69 model()

```
const Model& smtrat::Module::model ( ) const [inline], [inherited]
```

Returns

The assignment of the current satisfiable result, if existent.

0.15.533.4.70 modelsDisjoint()

```
bool smtrat::Module::modelsDisjoint (
    const Model & _modelA,
    const Model & _modelB ) [static], [protected], [inherited]
```

Checks whether there is no variable assigned by both models.

Parameters

_modelA	The first model to check for.
_modelB	The second model to check for.

Returns

true, if there is no variable assigned by both models; false, otherwise.

0.15.533.4.71 moduleName()

```
template<typename Settings >
std::string smtrat::UFCegarModule< Settings >::moduleName ( ) const [inline], [virtual]
```

Returns

The name of the given module type as name.

Reimplemented from smtrat::Module.

0.15.533.4.72 objective()

```
carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]
```

0.15.533.4.73 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.533.4.74 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.533.4.75 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.533.4.76 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.533.4.77 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.533.4.78 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.533.4.79 printAllModels()

`void smtrat::Module::printAllModels ( std::ostream & _out = std::cout ) [inherited]`

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.533.4.80 printInfeasibleSubsets()

`void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.533.4.81 printModel()

`void smtrat::Module::printModel ( std::ostream & _out = std::cout ) const [inherited]`

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

_out	The stream to print the assignment on.

0.15.533.4.82 printPassedFormula()

`void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.533.4.83 printReceivedFormula()

`void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.533.4.84 probablyLooping()

`bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.533.4.85 receivedFormulaChecked() `void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.533.4.86 receivedFormulasAsInfeasibleSubset() `void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

0.15.533.4.87 receivedVariable() `bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

0.15.533.4.88 remove() `void smtrat::Module::remove ( ModuleInput::const_iterator _subformula ) [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.
However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

Parameters

_subformula	The sub formula of the received formula to remove.

Reimplemented in smtrat::PModule.

0.15.533.4.89 removeCore() [1/2] `virtual void smtrat::Module::removeCore ( [[maybe_unused] ] 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.

0.15.533.4.90 removeCore() [2/2] `template<typename Settings > void smtrat::UFCegarModule< Settings >::removeCore ( ModuleInput::const_iterator _subformula )`

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.

0.15.533.4.91 removeOrigin() `std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩Origin (`

```
ModuleInput::iterator _formula,
const FormulaT & _origin )  [inline], [protected], [inherited]
```

0.15.533.4.92 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins (
ModuleInput::iterator _formula,
const std::shared_ptr< std::vector< FormulaT >> & _origins )  [inline], [protected], [inherited]
```

0.15.533.4.93 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.533.4.94 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]
```

Returns

A reference to the formula passed to this module.

0.15.533.4.95 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( )  [inline], [protected], [virtual], [inherited]
```

Reimplemented in smtrat::PModule.

0.15.533.4.96 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.

0.15.533.4.97 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.

0.15.533.4.98 setThreadPriority()

`void smtrat::Module::setThreadPriority ( thread_priority _threadPriority ) [inline], [inherited]`

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.533.4.99 solverState()

`Answer smtrat::Module::solverState ( ) const [inline], [inherited]`

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.533.4.100 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 !=.

0.15.533.4.101 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.533.4.102 updateAllModels()

`void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented in smtrat::SATModule< Settings >.

0.15.533.4.103 updateLemmas()

`void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.533.4.104 updateModel() `template<typename Settings >`
`void smtrat::UFCegarModule< Settings >::updateModel ( ) const [virtual]`

Updates the current assignment into the model.
Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.
Reimplemented from smtrat::Module.

0.15.533.4.105 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).

0.15.533.5 Field Documentation

0.15.533.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.533.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.533.5.3 mBackendsFoundAnswer `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.533.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.533.5.5 mFinalCheck `bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.533.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.533.5.7 mFirstSubformulaToPass `ModuleInput::iterator smtrat::Module::mFirstSubformulaTo↩ Pass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.533.5.8 mFirstUncheckedReceivedSubformula `ModuleInput::const_iterator smtrat::Module↩ ::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.533.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.533.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.533.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.533.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.533.5.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std↩::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.533.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.533.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.533.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.533.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.533.5.18 mObjectiveVariable

`carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.533.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplitting↩Variables [static], [inherited]`

Reusable splitting variables.

0.15.533.5.20 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.533.5.21 mSmallerMusesCheckCounter

`unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.533.5.22 mSolverState

`std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.533.5.23 mTheoryPropagations

`std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

0.15.533.5.24 mUsedBackends

`std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.533.5.25 mVariableCounters

`std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.534 smtrat::expression::UnaryExpression Struct Reference

`#include <ExpressionContent.h>`

Public Member Functions

- UnaryExpression (UnaryType _type, Expression &&_expression)

Data Fields

- UnaryType type
- Expression expression

0.15.534.1 Constructor & Destructor Documentation

0.15.534.1.1 UnaryExpression()

```
smtrat::expression::UnaryExpression::UnaryExpression (
    UnaryType _type,
    Expression && _expression ) [inline]
```

0.15.534.2 Field Documentation

0.15.534.2.1 expression

`Expression smtrat::expression::UnaryExpression::expression`

0.15.534.2.2 type

`UnaryType smtrat::expression::UnaryExpression::type`

0.15.535 smtrat::cad::debug::UnifiedData Struct Reference

`#include <TikzHistoryPrinter.h>`

Public Member Functions

- void add (std::size_t step, const std::string &data)
- bool showsOn (std::size_t step) const
- const std::string & data (std::size_t step) const

Data Fields

- std::vector< std::optional< std::string > > steps

0.15.535.1 Member Function Documentation

0.15.535.1.1 add()

```
void smtrat::cad::debug::UnifiedData::add (
    std::size_t step,
    const std::string & data ) [inline]
```

0.15.535.1.2 data()

```
const std::string& smtrat::cad::debug::UnifiedData::data (
    std::size_t step ) const [inline]
```

0.15.535.1.3 showsOn()

```
bool smtrat::cad::debug::UnifiedData::showsOn (
    std::size_t step ) const [inline]
```

0.15.535.2 Field Documentation

0.15.535.2.1 steps

```
std::vector<std::optional<std::string> > smtrat::cad::debug::UnifiedData↩
::steps
```

0.15.536 smtrat::parser::types::UninterpretedTheory Struct Reference

Types of the theory of equalities and uninterpreted functions.

```
#include <TheoryTypes.h>
```

Public Types

- typedef mpl::vector< carl::UTerm > ConstTypes
- typedef mpl::vector< carl::UVariable > VariableTypes
- typedef mpl::vector< carl::UTerm > ExpressionTypes
- typedef mpl::vector< carl::UTerm, carl::UVariable > TermTypes
- typedef carl::mpl_variant_of< TermTypes >::type TermType

0.15.536.1 Detailed Description

Types of the theory of equalities and uninterpreted functions.

0.15.536.2 Member Typedef Documentation

0.15.536.2.1 ConstTypes

```
typedef mpl::vector<carl::UTerm> smtrat::parser::types::UninterpretedTheory::ConstTyp
```

0.15.536.2.2 ExpressionTypes `typedef mpl::vector<carl::UTerm> smtrat::parser::types::UninterpretedTheory::Exp`

0.15.536.2.3 TermType `typedef carl::mpl_variant_of<TermTypes>::type smtrat::parser::types::UninterpretedTheo`

0.15.536.2.4 TermTypes `typedef mpl::vector<carl::UTerm, carl::UVariable> smtrat::parser::types::Uninterprete`

0.15.536.2.5 VariableTypes `typedef mpl::vector<carl::UVariable> smtrat::parser::types::UninterpretedTheory::Va`

0.15.537 smtrat::parser::UninterpretedTheory Struct Reference

Implements the theory of equalities and uninterpreted functions.
`#include <Uninterpreted.h>`

Public Member Functions

- UninterpretedTheory (ParserState *state)
- FormulaT coupleBooleanVariables (carl::Variable v, carl::UVariable uv)
- bool declareVariable (const std::string &name, const carl::Sort &sort, types::VariableType &result, TheoryError &errors)

  Declare a new variable with the given name and the given sort.
- bool handleITE (const FormulaT &ifterm, const types::TermType &thenterm, const types::TermType &elseterm, types::TermType &result, TheoryError &errors)

  Resolve an if-then-else operator.
- bool handleFunctionInstantiation (const carl::UninterpretedFunction &f, const std::vector< types::TermType > &arguments, types::TermType &result, TheoryError &errors)
- bool handleDistinct (const std::vector< types::TermType > &, types::TermType &, TheoryError &errors)

  Resolve a distinct operator.
- bool functionCall (const Identifier &identifier, const std::vector< types::TermType > &arguments, types::TermType &result, TheoryError &errors)

  Resolve another unknown function call.
- virtual bool resolveSymbol (const Identifier &, types::TermType &, TheoryError &errors)

  Resolve a symbol that was not declared within the ParserState.
- template<typename T , typename Builder >
  FormulaT expandDistinct (const std::vector< T > &values, const Builder &neqBuilder)
- virtual bool instantiate (const types::VariableType &, const types::TermType &, types::TermType &, TheoryError &errors)

  Instantiate a variable within a term.
- virtual bool refreshVariable (const types::VariableType &, types::VariableType &, TheoryError &errors)

Static Public Member Functions

- static void addSimpleSorts (qi::symbols< char, carl::Sort > &)

  Initialize the global symbol table for simple sorts.
- static void addConstants (qi::symbols< char, types::ConstType > &)

  Initialize the global symbol table for constants.

Data Fields

- std::map< types::TermType, carl::UTerm > mInstantiatedArguments
- carl::Sort mBoolSort
- carl::UVariable mTrue
- carl::UVariable mFalse
- ParserState * state

0.15.537.1 Detailed Description

Implements the theory of equalities and uninterpreted functions.

0.15.537.2 Constructor & Destructor Documentation

0.15.537.2.1 UninterpretedTheory()

```
smtrat::parser::UninterpretedTheory::UninterpretedTheory (
    ParserState * state )
```

0.15.537.3 Member Function Documentation

0.15.537.3.1 addConstants()

```
static void smtrat::parser::AbstractTheory::addConstants (
    qi::symbols< char, types::ConstType > & ) [inline], [static], [inherited]
```

Initialize the global symbol table for constants.

0.15.537.3.2 addSimpleSorts()

```
static void smtrat::parser::AbstractTheory::addSimpleSorts (
    qi::symbols< char, carl::Sort > & ) [inline], [static], [inherited]
```

Initialize the global symbol table for simple sorts.

0.15.537.3.3 coupleBooleanVariables()

```
FormulaT smtrat::parser::UninterpretedTheory::coupleBooleanVariables (
    carl::Variable v,
    carl::UVariable uv ) [inline]
```

0.15.537.3.4 declareVariable()

```
bool smtrat::parser::UninterpretedTheory::declareVariable (
    const std::string & ,
    const carl::Sort & ,
    types::VariableType & ,
    TheoryError & errors ) [virtual]
```

Declare a new variable with the given name and the given sort.
Reimplemented from smtrat::parser::AbstractTheory.

0.15.537.3.5 expandDistinct()

```
template<typename T , typename Builder >
FormulaT smtrat::parser::AbstractTheory::expandDistinct (
    const std::vector< T > & values,
    const Builder & neqBuilder ) [inline], [inherited]
```

0.15.537.3.6 functionCall()

```
bool smtrat::parser::UninterpretedTheory::functionCall (
    const Identifier & ,
    const std::vector< types::TermType > & ,
    types::TermType & ,
    TheoryError & errors ) [virtual]
```

Resolve another unknown function call.
Reimplemented from smtrat::parser::AbstractTheory.

0.15.537.3.7 handleDistinct()

```
bool smtrat::parser::UninterpretedTheory::handleDistinct (
    const std::vector< types::TermType > & ,
    types::TermType & ,
    TheoryError & errors )  [virtual]
```

Resolve a distinct operator.
Reimplemented from smtrat::parser::AbstractTheory.

0.15.537.3.8 handleFunctionInstantiation()

```
bool smtrat::parser::UninterpretedTheory::handle↩
FunctionInstantiation (
    const carl::UninterpretedFunction & f,
    const std::vector< types::TermType > & arguments,
    types::TermType & result,
    TheoryError & errors )
```

0.15.537.3.9 handleITE()

```
bool smtrat::parser::UninterpretedTheory::handleITE (
    const FormulaT & ,
    const types::TermType & ,
    const types::TermType & ,
    types::TermType & ,
    TheoryError & errors )  [virtual]
```

Resolve an if-then-else operator.
Reimplemented from smtrat::parser::AbstractTheory.

0.15.537.3.10 instantiate()

```
virtual bool smtrat::parser::AbstractTheory::instantiate (
    const types::VariableType & ,
    const types::TermType & ,
    types::TermType & ,
    TheoryError & errors )  [inline], [virtual], [inherited]
```

Instantiate a variable within a term.
Reimplemented in smtrat::parser::BitvectorTheory, and smtrat::parser::ArithmeticTheory.

0.15.537.3.11 refreshVariable()

```
virtual bool smtrat::parser::AbstractTheory::refreshVariable (
    const types::VariableType & ,
    types::VariableType & ,
    TheoryError & errors )  [inline], [virtual], [inherited]
```

Reimplemented in smtrat::parser::BitvectorTheory.

0.15.537.3.12 resolveSymbol()

```
virtual bool smtrat::parser::AbstractTheory::resolveSymbol (
    const Identifier & ,
    types::TermType & ,
    TheoryError & errors )  [inline], [virtual], [inherited]
```

Resolve a symbol that was not declared within the ParserState.
This might be a symbol that actually uses indices, for example bitvector constants.
Reimplemented in smtrat::parser::BitvectorTheory.

0.15.537.4 Field Documentation

0.15.537.4.1 mBoolSort

```
carl::Sort smtrat::parser::UninterpretedTheory::mBoolSort
```

0.15.537.4.2 mFalse `carl::UVariable smtrat::parser::UninterpretedTheory::mFalse`

0.15.537.4.3 mInstantiatedArguments `std::map<types::TermType, carl::UTerm> smtrat::parser::↩ UninterpretedTheory::mInstantiatedArguments`

0.15.537.4.4 mTrue `carl::UVariable smtrat::parser::UninterpretedTheory::mTrue`

0.15.537.4.5 state `ParserState* smtrat::parser::AbstractTheory::state [inherited]`

0.15.538 smtrat::UnionFindInterface< T, Implementation > Struct Template Reference

`#include <UnionFind.h>`

Public Types

- using Value = typename Implementation::Value
- using Representative = typename Implementation::Representative
- using TranslateMap = std::unordered_map< T, Value >

Public Member Functions

- UnionFindInterface (TranslateMap &translate)
- Value insert_value (T const &var) noexcept
- template<typename Container >
  void init (Container const &cont) noexcept
- bool has_variable (T const &var) noexcept
- auto find (T const &val) noexcept -> Representative
- void merge (T const &a, T const &b) noexcept
- void backtrack (T const &a, T const &b) noexcept

0.15.538.1 Member Typedef Documentation

0.15.538.1.1 Representative

```
template<typename T , typename Implementation >
using smtrat::UnionFindInterface< T, Implementation >::Representative = typename Implementation↩
::Representative
```

0.15.538.1.2 TranslateMap

```
template<typename T , typename Implementation >
using smtrat::UnionFindInterface< T, Implementation >::TranslateMap = std::unordered_map<T,
Value>
```

0.15.538.1.3 Value

```
template<typename T , typename Implementation >
using smtrat::UnionFindInterface< T, Implementation >::Value = typename Implementation::Value
```

0.15.538.2 Constructor & Destructor Documentation

0.15.538.2.1 UnionFindInterface()

```
template<typename T , typename Implementation >
smtrat::UnionFindInterface< T, Implementation >::UnionFindInterface (
            TranslateMap & translate ) [inline]
```

0.15.538.3 Member Function Documentation

0.15.538.3.1 backtrack()

```
template<typename T , typename Implementation >
void smtrat::UnionFindInterface< T, Implementation >::backtrack (
            T const & a,
            T const & b )  [inline], [noexcept]
```

0.15.538.3.2 find()

```
template<typename T , typename Implementation >
auto smtrat::UnionFindInterface< T, Implementation >::find (
            T const & val ) -> Representative   [inline], [noexcept]
```

0.15.538.3.3 has_variable()

```
template<typename T , typename Implementation >
bool smtrat::UnionFindInterface< T, Implementation >::has_variable (
            T const & var )  [inline], [noexcept]
```

0.15.538.3.4 init()

```
template<typename T , typename Implementation >
template<typename Container >
void smtrat::UnionFindInterface< T, Implementation >::init (
            Container const & cont )  [inline], [noexcept]
```

0.15.538.3.5 insert_value()

```
template<typename T , typename Implementation >
Value smtrat::UnionFindInterface< T, Implementation >::insert_value (
            T const & var )  [inline], [noexcept]
```

0.15.538.3.6 merge()

```
template<typename T , typename Implementation >
void smtrat::UnionFindInterface< T, Implementation >::merge (
            T const & a,
            T const & b )  [inline], [noexcept]
```

0.15.539 smtrat::UnionFindModule< Settings > Class Template Reference

`#include <UnionFindModule.h>`

Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- UnionFindModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=nullptr)
- ~UnionFindModule ()
- bool informCore (const FormulaT &_constraint)

  Informs the module about the given constraint.
- bool addCore (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- void removeCore (ModuleInput::const_iterator _subformula)

  Removes the subformula of the received formula at the given position to the considered ones of this module.

- void updateModel () const

  Updates the current assignment into the model.
- Answer checkCore ()

  Checks the received formula for consistency.
- bool inform (const FormulaT &_constraint)

  Informs the module about the given constraint.
- void deinform (const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual void init ()

  Informs all backends about the so far encountered constraints, which have not yet been communicated.
- bool add (ModuleInput::const_iterator _subformula)

  The module has to take the given sub-formula of the received formula into account.
- virtual Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_↩ VARIABLE)

  Checks the received formula for consistency.
- virtual void remove (ModuleInput::const_iterator _subformula)

  Removes everything related to the given sub-formula of the received formula.
- virtual void updateAllModels ()

  Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
- virtual std::list< std::vector< carl::Variable > > getModelEqualities () const

  Partition the variables from the current model into equivalence classes according to their assigned value.
- unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const
- virtual unsigned currentlySatisfied (const FormulaT &) const
- bool receivedVariable (carl::Variable::Arg _var) const
- Answer solverState () const
- std::size_t id () const
- void setId (std::size_t _id)

  Sets this modules unique ID to identify itself.
- thread_priority threadPriority () const
- void setThreadPriority (thread_priority _threadPriority)

  Sets the priority of this module to get a thread for running its check procedure.
- const ModuleInput * pReceivedFormula () const
- const ModuleInput & rReceivedFormula () const
- const ModuleInput * pPassedFormula () const
- const ModuleInput & rPassedFormula () const
- const Model & model () const
- const std::vector< Model > & allModels () const
- const std::vector< FormulaSetT > & infeasibleSubsets () const
- const std::vector< Module * > & usedBackends () const
- const carl::FastSet< FormulaT > & constraintsToInform () const
- const carl::FastSet< FormulaT > & informedConstraints () const
- void addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))

  Stores a lemma being a valid formula.
- bool hasLemmas ()

  Checks whether this module or any of its backends provides any lemmas.
- void clearLemmas ()

  Deletes all yet found lemmas.
- const std::vector< Lemma > & lemmas () const
- ModuleInput::const_iterator firstUncheckedReceivedSubformula () const
- ModuleInput::const_iterator firstSubformulaToPass () const
- void receivedFormulaChecked ()

Notifies that the received formulas has been checked.

- const 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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.
- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()
- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)
- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

Copies the infeasible subsets of the passed formula.

- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const
- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

  This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.539.1 Member Typedef Documentation

0.15.539.1.1 SettingsType

```
template<typename Settings >
typedef Settings smtrat::UnionFindModule< Settings >::SettingsType
```

0.15.539.2 Member Enumeration Documentation

0.15.539.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

NORMAL	
PERMANENT	

0.15.539.3 Constructor & Destructor Documentation

0.15.539.3.1 UnionFindModule() `template<typename Settings >`

```
smtrat::UnionFindModule< Settings >::UnionFindModule (
            const ModuleInput * _formula,
            Conditionals & _conditionals,
            Manager * _manager = nullptr )
```

0.15.539.3.2 ~UnionFindModule() `template<typename Settings >`

```
smtrat::UnionFindModule< Settings >::~UnionFindModule ( )
```

0.15.539.4 Member Function Documentation

0.15.539.4.1 add() `bool smtrat::Module::add (`

```
            ModuleInput::const_iterator _subformula )  [inherited]
```

The module has to take the given sub-formula of the received formula into account.

Parameters

_subformula	The sub-formula to take additionally into account.

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

0.15.539.4.2 addConstraintToInform() `void smtrat::Module::addConstraintToInform (`

```
            const FormulaT & _constraint )  [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

0.15.539.4.3 addCore() [1/2] `virtual bool smtrat::Module::addCore (`

```
[[maybe_unused] ] 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.

0.15.539.4.4 addCore() [2/2]

```
template<typename Settings >
bool smtrat::UnionFindModule< Settings >::addCore (
            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.

0.15.539.4.5 addInformationRelevantFormula()

```
void smtrat::Module::addInformationRelevantFormula
(
            const FormulaT & formula )  [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.539.4.6 addLemma()

```
void smtrat::Module::addLemma (
            const FormulaT & _lemma,
            const LemmaType & _lt = LemmaType::NORMAL,
            const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) )  [inline],
[inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.539.4.7 addOrigin()

```
void smtrat::Module::addOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.539.4.8 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.539.4.9 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat↩
::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.539.4.10 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.539.4.11 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.

0.15.539.4.12 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const [inline],
[inherited]
```

Returns

All satisfying assignments, if existent.

0.15.539.4.13 anAnswerFound()

```
bool smtrat::Module::anAnswerFound ( ) const [inline], [protected],
[inherited]
```

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.539.4.14 answerFound()

```
const smtrat::Conditionals& smtrat::Module::answerFound ( ) const
[inline], [inherited]
```

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.539.4.15 backendsModel()

```
const Model & smtrat::Module::backendsModel ( ) const [protected],
[inherited]
```

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.539.4.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline],
[protected], [inherited]
```

0.15.539.4.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.539.4.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline],
[protected], [inherited]
```

0.15.539.4.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.539.4.20 check()

```
Answer smtrat::Module::check (
    bool _final = false,
    bool _full = true,
    carl::Variable _objective = carl::Variable::NO_VARIABLE ) [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.539.4.21 checkCore()

```
template<typename Settings >
Answer smtrat::UnionFindModule< Settings >::checkCore ( ) [virtual]
```

Checks the received formula for consistency.

Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from smtrat::Module.

0.15.539.4.22 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.

0.15.539.4.23 checkModel()

```
unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]
```

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.539.4.24 cleanModel() `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.539.4.25 clearLemmas() `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

0.15.539.4.26 clearModel() `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

0.15.539.4.27 clearModels() `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

0.15.539.4.28 clearPassedFormula() `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

0.15.539.4.29 collectOrigins() [1/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

0.15.539.4.30 collectOrigins() [2/2] `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inherited]`

Collects the formulas in the given formula, which are part of the received formula.
If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.539.4.31 collectTheoryPropagations() `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

0.15.539.4.32 constraintsToInform() `const carl::FastSet<FormulaT>& smtrat::Module::constraints↩ ToInform ( ) const [inline], [inherited]`

Returns

The constraints which the backends must be informed about.

0.15.539.4.33 currentlySatisfied()

`virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.539.4.34 currentlySatisfiedByBackend()

`unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const [inherited]`

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.539.4.35 deinform()

`void smtrat::Module::deinform ( const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.539.4.36 deinformCore()

`virtual void smtrat::Module::deinformCore ( [[maybe_unused] ] 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.

0.15.539.4.37 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

0.15.539.4.38 eraseSubformulaFromPassedFormula()

```
ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (
            ModuleInput::iterator _subformula,
            bool _ignoreOrigins = false ) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_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 >.

0.15.539.4.39 excludeNotReceivedVariablesFromModel()

```
void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]
```

Excludes all variables from the current model, which do not occur in the received formula.

0.15.539.4.40 findBestOrigin()

```
std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
            const std::vector< FormulaT > & _origins ) const [protected], [inherited]
```

Parameters

_origins	

Returns

0.15.539.4.41 firstSubformulaToPass()

```
ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]
```

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.539.4.42 firstUncheckedReceivedSubformula()

```
ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]
```

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.539.4.43 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.539.4.44 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasible↩Subset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.539.4.45 getBackendsAllModels()

`void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.539.4.46 getBackendsModel()

`void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.539.4.47 getInfeasibleSubsets() [1/2]

`void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.539.4.48 getInfeasibleSubsets() [2/2]

`std::vector< FormulaSetT > smtrat::Module::get↩InfeasibleSubsets ( 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.

0.15.539.4.49 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module↩::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.539.4.50 getModelEqualities()

`std::list< std::vector< carl::Variable > > smtrat::Module↩::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.539.4.51 getOrigins() [1/3]

`void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.539.4.52 getOrigins() [2/3]

`void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

Parameters

_formula	
_origins	

0.15.539.4.53 getOrigins() [3/3]

`const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]`

Gets the origins of the passed formula at the given position.

Parameters

_formula	The position of a formula in the passed formulas.

Returns

The origins of the passed formula at the given position.

0.15.539.4.54 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.

0.15.539.4.55 hasLemmas()

`bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

0.15.539.4.56 hasValidInfeasibleSubset()

`bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]`

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.539.4.57 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.539.4.58 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasible↩ Subsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.539.4.59 inform() `bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.
It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

Parameters

_constraint	The constraint to inform about.

Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

0.15.539.4.60 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.

0.15.539.4.61 informCore() [1/2] `virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.539.4.62 informCore() [2/2]

```
template<typename Settings >
bool smtrat::UnionFindModule< Settings >::informCore (
            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.

0.15.539.4.63 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.

0.15.539.4.64 init()

```
void smtrat::Module::init ( ) [virtual], [inherited]
```

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not be called twice and only before the first runBackends call.
Reimplemented in smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntBlastModule< Settings >, smtrat::FPPModule< Settings >, smtrat::CurryModule< Settings >, and smtrat::BVModule< Settings >.

0.15.539.4.65 is_minimizing()

```
bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]
```

0.15.539.4.66 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.

0.15.539.4.67 isPreprocessor()

```
bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]
```

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.539.4.68 lemmas()

```
const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline],
[inherited]
```

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.539.4.69 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.

0.15.539.4.70 model()

```
const Model& smtrat::Module::model ( ) const [inline], [inherited]
```

Returns

The assignment of the current satisfiable result, if existent.

0.15.539.4.71 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.

0.15.539.4.72 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.

0.15.539.4.73 objective()

```
carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]
```

0.15.539.4.74 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.539.4.75 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.539.4.76 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.539.4.77 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.539.4.78 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.539.4.79 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.539.4.80 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.

0.15.539.4.81 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
const std::string & _initiation = "***" ) const [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.539.4.82 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.

0.15.539.4.83 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.539.4.84 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.539.4.85 probablyLooping()

```
bool smtrat::Module::probablyLooping (
const typename Poly::PolyType & _branchingPolynomial,
const Rational & _branchingValue ) const [protected], [inherited]
```

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.539.4.86 receivedFormulaChecked() `void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.539.4.87 receivedFormulasAsInfeasibleSubset() `void smtrat::Module::receivedFormulasAs↩InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

0.15.539.4.88 receivedVariable() `bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

0.15.539.4.89 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.

0.15.539.4.90 removeCore() [1/2] `virtual void smtrat::Module::removeCore ( [[maybe_unused] ] 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.

0.15.539.4.91 removeCore() [2/2] `template<typename Settings > void smtrat::UnionFindModule< Settings >::removeCore ( ModuleInput::const_iterator _subformula )`

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.

0.15.539.4.92 removeOrigin() `std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩Origin (`

```
ModuleInput::iterator _formula,
const FormulaT & _origin )  [inline], [protected], [inherited]
```

0.15.539.4.93 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins (
ModuleInput::iterator _formula,
const std::shared_ptr< std::vector< FormulaT >> & _origins )  [inline], [protected], [inherited]
```

0.15.539.4.94 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.539.4.95 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]
```

Returns

A reference to the formula passed to this module.

0.15.539.4.96 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( )  [inline], [protected], [virtual], [inherited]
```

Reimplemented in smtrat::PModule.

0.15.539.4.97 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.

0.15.539.4.98 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.

0.15.539.4.99 setThreadPriority()

`void smtrat::Module::setThreadPriority ( thread_priority _threadPriority ) [inline], [inherited]`

Sets the priority of this module to get a thread for running its check procedure.

Parameters

_threadPriority	The priority to set this module's thread priority to.

0.15.539.4.100 solverState()

`Answer smtrat::Module::solverState ( ) const [inline], [inherited]`

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.539.4.101 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 !=.

0.15.539.4.102 threadPriority()

`thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.539.4.103 updateAllModels()

`void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
Reimplemented in smtrat::SATModule< Settings >.

0.15.539.4.104 updateLemmas()

`void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.539.4.105 updateModel() `template<typename Settings > void smtrat::UnionFindModule< Settings >::updateModel ( ) const [virtual]`

Updates the current assignment into the model.
Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.
Reimplemented from smtrat::Module.

0.15.539.4.106 usedBackends() `const std::vector<Module*>& smtrat::Module::usedBackends ( ) const [inline], [inherited]`

Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.539.5 Field Documentation

0.15.539.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.539.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.539.5.3 mBackendsFoundAnswer `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.539.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.539.5.5 mFinalCheck `bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.539.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.539.5.7 mFirstSubformulaToPass `ModuleInput::iterator smtrat::Module::mFirstSubformulaTo↩ Pass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.539.5.8 mFirstUncheckedReceivedSubformula `ModuleInput::const_iterator smtrat::Module↩ ::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.539.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.539.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.539.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.539.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.539.5.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std↩::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.539.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.539.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.539.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.539.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.539.5.18 mObjectiveVariable

`carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.539.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplitting↩Variables [static], [inherited]`

Reusable splitting variables.

0.15.539.5.20 mpManager

`Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.539.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.539.5.22 mSolverState `std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.539.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

0.15.539.5.24 mUsedBackends `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.539.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.540 smtrat::SATModule< Settings >::UnorderedClauseLookup::UnorderedClauseHasher Struct Reference

`#include <SATModule.h>`

Public Member Functions

- std::size_t operator() (const std::vector< Minisat::Lit > &cl) const

0.15.540.1 Member Function Documentation

0.15.540.1.1 operator()() `template<class Settings > std::size_t smtrat::SATModule< Settings >::UnorderedClauseLookup::UnorderedClauseHasher↩ ::operator() ( const std::vector< Minisat::Lit > & cl ) const [inline]`

0.15.541 smtrat::UnsatCore< Solver, Strategy > Class Template Reference

`#include <UnsatCore.h>`

Public Member Functions

- UnsatCore (Solver &s)
- void add_annotated_name (const FormulaT &formula, const std::string &name)
- void remove_annotated_name (const FormulaT &formula)
- void reset ()
- bool active () const
- std::pair< Answer, FormulasT > compute_core (const FormulasT &formulas)
- std::pair< Answer, std::vector< std::string > > compute ()

0.15.541.1 Constructor & Destructor Documentation

0.15.541.1.1 UnsatCore()

```
template<typename Solver , UnsatCoreStrategy Strategy>
smtrat::UnsatCore< Solver, Strategy >::UnsatCore (
            Solver & s )  [inline]
```

0.15.541.2 Member Function Documentation

0.15.541.2.1 active()

```
template<typename Solver , UnsatCoreStrategy Strategy>
bool smtrat::UnsatCore< Solver, Strategy >::active ( ) const  [inline]
```

0.15.541.2.2 add_annotated_name()

```
template<typename Solver , UnsatCoreStrategy Strategy>
void smtrat::UnsatCore< Solver, Strategy >::add_annotated_name (
            const FormulaT & formula,
            const std::string & name )  [inline]
```

0.15.541.2.3 compute()

```
template<typename Solver , UnsatCoreStrategy Strategy>
std::pair<Answer,std::vector<std::string> > smtrat::UnsatCore< Solver, Strategy >::compute (
)  [inline]
```

0.15.541.2.4 compute_core()

```
template<typename Solver , UnsatCoreStrategy Strategy>
std::pair<Answer,FormulasT> smtrat::UnsatCore< Solver, Strategy >::compute_core (
            const FormulasT & formulas )  [inline]
```

0.15.541.2.5 remove_annotated_name()

```
template<typename Solver , UnsatCoreStrategy Strategy>
void smtrat::UnsatCore< Solver, Strategy >::remove_annotated_name (
            const FormulaT & formula )  [inline]
```

0.15.541.2.6 reset()

```
template<typename Solver , UnsatCoreStrategy Strategy>
void smtrat::UnsatCore< Solver, Strategy >::reset ( )  [inline]
```

0.15.542 smtrat::unsatcore::UnsatCoreBackend< Solver, Strategy > Class Template Reference

```
#include <UnsatCore.h>
```

0.15.543 smtrat::unsatcore::UnsatCoreBackend< Solver, UnsatCoreStrategy::ModelExclusion > Class Template Reference

Implements an easy strategy to obtain an unsatisfiable core.

```
#include <UnsatCore_ModelExclusion.h>
```

Public Member Functions

- UnsatCoreBackend (Solver &s, const FormulasT &fs)
- void processAssignment ()
- Answer compute ()
- std::pair< Answer, FormulasT > run ()

0.15.543.1 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)
-

0.15.543.2 Constructor & Destructor Documentation

0.15.543.2.1 UnsatCoreBackend() template<typename Solver >

```
smtrat::unsatcore::UnsatCoreBackend< Solver, UnsatCoreStrategy::ModelExclusion >::UnsatCoreBackend
(
            Solver & s,
            const FormulasT & fs )  [inline]
```

0.15.543.3 Member Function Documentation

0.15.543.3.1 compute() template<typename Solver >

```
Answer smtrat::unsatcore::UnsatCoreBackend< Solver, UnsatCoreStrategy::ModelExclusion >↩
::compute ( )  [inline]
```

0.15.543.3.2 processAssignment() template<typename Solver >

```
void smtrat::unsatcore::UnsatCoreBackend< Solver, UnsatCoreStrategy::ModelExclusion >::process↩
Assignment ( )  [inline]
```

0.15.543.3.3 run() template<typename Solver >

```
std::pair<Answer, FormulasT> smtrat::unsatcore::UnsatCoreBackend< Solver, UnsatCoreStrategy::ModelExclusion
>::run ( )  [inline]
```

0.15.544 smtrat::vs::unsignedTripleCmp Struct Reference

```
#include <State.h>
```

Public Member Functions

- bool operator() (UnsignedTriple n1, UnsignedTriple n2) const

0.15.544.1 Member Function Documentation

0.15.544.1.1 operator()() bool smtrat::vs::unsignedTripleCmp::operator() (

```
            UnsignedTriple n1,
            UnsignedTriple n2 ) const  [inline]
```

0.15.545 smtrat::parser::UserFunctionInstantiator Struct Reference

`#include <FunctionInstantiator.h>`

Public Member Functions

- UserFunctionInstantiator (const std::vector< types::VariableType > &arguments, const carl::Sort &sort, const types::TermType &definition, const std::set< types::VariableType > &auxiliaries)
- template<typename T >
  bool convert (const std::vector< types::TermType > &from, std::vector< T > &to) const
- template<typename T >
  bool convert (const std::vector< types::TermType > &from, std::vector< T > &to, TheoryError &errors) const
- virtual bool operator() (const std::vector< types::TermType > &, types::TermType &, TheoryError &errors) const

Data Fields

- std::vector< types::VariableType > arguments
- carl::Sort sort
- types::TermType definition
- std::set< types::VariableType > auxiliaries
- FormulasT globalFormulas

0.15.545.1 Constructor & Destructor Documentation

0.15.545.1.1 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]
```

0.15.545.2 Member Function Documentation

0.15.545.2.1 convert() [1/2]

```
template<typename T >
bool smtrat::parser::FunctionInstantiator::convert (
            const std::vector< types::TermType > & from,
            std::vector< T > & to ) const  [inline], [inherited]
```

0.15.545.2.2 convert() [2/2]

```
template<typename T >
bool smtrat::parser::FunctionInstantiator::convert (
            const std::vector< types::TermType > & from,
            std::vector< T > & to,
            TheoryError & errors ) const  [inline], [inherited]
```

0.15.545.2.3 operator()()

```
virtual bool smtrat::parser::FunctionInstantiator::operator() (
            const std::vector< types::TermType > & ,
            types::TermType & ,
            TheoryError & errors ) const  [inline], [virtual], [inherited]
```

0.15.545.3 Field Documentation

0.15.545.3.1 arguments `std::vector<types::VariableType> smtrat::parser::UserFunctionInstantiator::arguments`

0.15.545.3.2 auxiliaries `std::set<types::VariableType> smtrat::parser::UserFunctionInstantiator::auxiliaries`

0.15.545.3.3 definition `types::TermType smtrat::parser::UserFunctionInstantiator::definition`

0.15.545.3.4 globalFormulas `FormulasT smtrat::parser::UserFunctionInstantiator::globalFormulas`

0.15.545.3.5 sort `carl::Sort smtrat::parser::UserFunctionInstantiator::sort`

0.15.546 smtrat::validation::ValidationCollector Class Reference

`#include <ValidationCollector.h>`

Public Member Functions

- ValidationPoint & get (const std::string &channel, const std::string &file, int line)
- const auto & points () const

0.15.546.1 Member Function Documentation

0.15.546.1.1 get()

```
ValidationPoint& smtrat::validation::ValidationCollector::get (
    const std::string & channel,
    const std::string & file,
    int line ) [inline]
```

0.15.546.1.2 points()

```
const auto& smtrat::validation::ValidationCollector::points ( ) const [inline]
```

0.15.547 smtrat::validation::ValidationPoint Class Reference

`#include <ValidationPoint.h>`

Public Member Functions

- void set_identifier (const std::string &channel, const std::string &file, int line)
- const auto & channel () const
- auto identifier () const
- const auto & formulas () const
- std::size_t add (const FormulaT &f, bool consistent, const std::string &name)
- std::size_t add (const FormulasT &fs, bool consistent, const std::string &name)
- std::size_t add (const FormulaSetT &fss, bool consistent, const std::string &name)
- std::size_t add (const ConstraintT &c, bool consistent, const std::string &name)
- std::size_t add (const ConstraintsT &cs, bool consistent, const std::string &name)

0.15.547.1 Member Function Documentation

0.15.547.1.1 add() [1/5]

```
std::size_t smtrat::validation::ValidationPoint::add (
    const ConstraintsT & cs,
    bool consistent,
    const std::string & name ) [inline]
```

0.15.547.1.2 add() [2/5]

```
std::size_t smtrat::validation::ValidationPoint::add (
    const ConstraintT & c,
    bool consistent,
    const std::string & name ) [inline]
```

0.15.547.1.3 add() [3/5]

```
std::size_t smtrat::validation::ValidationPoint::add (
    const FormulaSetT & fss,
    bool consistent,
    const std::string & name ) [inline]
```

0.15.547.1.4 add() [4/5]

```
std::size_t smtrat::validation::ValidationPoint::add (
    const FormulasT & fs,
    bool consistent,
    const std::string & name ) [inline]
```

0.15.547.1.5 add() [5/5]

```
std::size_t smtrat::validation::ValidationPoint::add (
    const FormulaT & f,
    bool consistent,
    const std::string & name ) [inline]
```

0.15.547.1.6 channel()

```
const auto& smtrat::validation::ValidationPoint::channel ( ) const [inline]
```

0.15.547.1.7 formulas()

```
const auto& smtrat::validation::ValidationPoint::formulas ( ) const
[inline]
```

0.15.547.1.8 identifier()

```
auto smtrat::validation::ValidationPoint::identifier ( ) const [inline]
```

0.15.547.1.9 set_identifier()

```
void smtrat::validation::ValidationPoint::set_identifier (
    const std::string & channel,
    const std::string & file,
    int line ) [inline]
```

0.15.548 smtrat::validation::ValidationPrinter< SOF > Struct Template Reference

```
#include <ValidationPrinter.h>
```

0.15.549 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

0.15.549.1 Member Function Documentation

0.15.549.1.1 channel_active() `bool smtrat::validation::ValidationSettings::channel_active ( const std::string & key ) const [inline]`

0.15.549.2 Field Documentation

0.15.549.2.1 channels `std::vector<std::string> smtrat::validation::ValidationSettings::channels`

0.15.549.2.2 export_as_smtlib `bool smtrat::validation::ValidationSettings::export_as_smtlib`

0.15.549.2.3 smtlib_filename `std::string smtrat::validation::ValidationSettings::smtlib_filename`

0.15.550 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 is_zero () const
- T sign () const

  Computes the sign: 0 if main & delta part are unequal 0.
- void print (std::ostream &_out=std::cout) const

0.15.550.1 Constructor & Destructor Documentation

0.15.550.1.1 Value() [1/4] template<class T >

```
smtrat::lra::Value< T >::Value ( )
```

0.15.550.1.2 Value() [2/4] template<class T >

```
smtrat::lra::Value< T >::Value (
            const T & _num )
```

Parameters

_num	

0.15.550.1.3 Value() [3/4] template<class T >

```
smtrat::lra::Value< T >::Value (
            const T & _num1,
            const T & _num2 )
```

Parameters

_num1	
_num2	

0.15.550.1.4 Value() [4/4] template<class T >

```
smtrat::lra::Value< T >::Value (
            const Value< T > & _orig )
```

Parameters

_orig	

0.15.550.1.5 ~Value() template<class T >

virtual smtrat::lra::Value< T >::~Value ( ) [virtual]

0.15.550.2 Member Function Documentation

0.15.550.2.1 abs() template<class T >

Value<T> smtrat::lra::Value< T >::abs ( ) const [inline]

0.15.550.2.2 abs_() template<class T >

const Value<T>& smtrat::lra::Value< T >::abs_ ( ) [inline]

0.15.550.2.3 deltaPart() template<class T >

const T& smtrat::lra::Value< T >::deltaPart ( ) const [inline]

Returns

0.15.550.2.4 is_zero() template<class T >

bool smtrat::lra::Value< T >::is_zero ( ) const [inline]

0.15.550.2.5 mainPart() template<class T >

const T& smtrat::lra::Value< T >::mainPart ( ) const [inline]

Returns

0.15.550.2.6 operator"!=() [1/2] template<class T >

bool smtrat::lra::Value< T >::operator!= (
const T & _a) const [inline]

0.15.550.2.7 operator"!=() [2/2] template<class T >

bool smtrat::lra::Value< T >::operator!= (
const Value< T > & _value) const [inline]

0.15.550.2.8 operator*() [1/2] template<class T >

Value<T> smtrat::lra::Value< T >::operator* (
const T & _a) const

Parameters

↩ _↩ a	

Returns

0.15.550.2.9 operator∗() [2/2] template<class T >

```
Value<T> smtrat::lra::Value< T >::operator* (
            const Value< T > & _value ) const
```

Parameters

_value	

0.15.550.2.10 operator∗=() [1/2] template<class T >

```
void smtrat::lra::Value< T >::operator*= (
            const T & _a )
```

Parameters

↩ ↩ *a*	

0.15.550.2.11 operator∗=() [2/2] template<class T >

```
void smtrat::lra::Value< T >::operator*= (
            const Value< T > & _value )
```

Parameters

_value	

0.15.550.2.12 operator+() template<class T >

```
Value<T> smtrat::lra::Value< T >::operator+ (
            const Value< T > & _value ) const
```

Parameters

_value	

Returns

0.15.550.2.13 operator+=() template<class T >

```
void smtrat::lra::Value< T >::operator+= (
            const Value< T > & _value )
```

Parameters

_value	

0.15.550.2.14 operator-()

```
template<class T >
Value<T> smtrat::lra::Value< T >::operator- (
            const Value< T > & _value ) const
```

Parameters

_value	

Returns

0.15.550.2.15 operator-=()

```
template<class T >
void smtrat::lra::Value< T >::operator-= (
            const Value< T > & _value )
```

Parameters

_value	

0.15.550.2.16 operator/()

```
template<class T >
Value<T> smtrat::lra::Value< T >::operator/ (
            const T & _a ) const
```

Parameters

↩ _↩ *a*	

Returns

0.15.550.2.17 operator/=()

```
template<class T >
void smtrat::lra::Value< T >::operator/= (
            const T & _a )
```

Parameters

↩ _↩ *a*	

0.15.550.2.18 operator<() [1/2] template<class T >

```
bool smtrat::lra::Value< T >::operator< (
            const T & _a ) const
```

Parameters

↩ _↩ *a*	

Returns

0.15.550.2.19 operator<() [2/2] template<class T >

```
bool smtrat::lra::Value< T >::operator< (
            const Value< T > & _value ) const
```

Parameters

_value	

Returns

0.15.550.2.20 operator<=() [1/2] template<class T >

```
bool smtrat::lra::Value< T >::operator<= (
            const T & _a ) const
```

Parameters

↩ _↩ *a*	

Returns

0.15.550.2.21 operator<=() [2/2] template<class T >

```
bool smtrat::lra::Value< T >::operator<= (
            const Value< T > & _value ) const
```

Parameters

_value	

Returns

0.15.550.2.22 operator=()

```
template<class T >
Value<T>& smtrat::lra::Value< T >::operator= (
            const Value< T > & _value )
```

Copy the content of the given value to this one.

Parameters

_value	The value to copy.

Returns

This value after copying.

0.15.550.2.23 operator==() [1/2]

```
template<class T >
bool smtrat::lra::Value< T >::operator== (
            const T & _a ) const
```

Parameters

↩ *_a*	

Returns

0.15.550.2.24 operator==() [2/2]

```
template<class T >
bool smtrat::lra::Value< T >::operator== (
            const Value< T > & _value ) const
```

Parameters

_value	

Returns

0.15.550.2.25 operator>() [1/2]

```
template<class T >
bool smtrat::lra::Value< T >::operator> (
            const T & _a ) const
```

Parameters

↩ *_a*	

Returns

0.15.550.2.26 operator>() [2/2] template<class T >

```
bool smtrat::lra::Value< T >::operator> (
            const Value< T > & _value ) const  [inline]
```

0.15.550.2.27 operator>=() [1/2] template<class T >

```
bool smtrat::lra::Value< T >::operator>= (
            const T & _a ) const
```

Parameters

↩ _↩ *a*	

Returns

0.15.550.2.28 operator>=() [2/2] template<class T >

```
bool smtrat::lra::Value< T >::operator>= (
            const Value< T > & _value ) const  [inline]
```

0.15.550.2.29 print() template<class T >

```
void smtrat::lra::Value< T >::print (
            std::ostream & _out = std::cout ) const
```

Parameters

_out	

0.15.550.2.30 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.

0.15.550.2.31 toString() template<class T >

```
const std::string smtrat::lra::Value< T >::toString ( ) const
```

Returns

0.15.551 smtrat::lra::Variable< T1, T2 > Class Template Reference

```
#include <Variable.h>
```

Public Member Functions

- Variable (size_t _position, const typename Poly::PolyType *_expression, ModuleInput::iterator _defaultBoundPosition, bool _isInteger, size_t _id)
- Variable (typename std::list< std::list< std::pair< Variable< T1, T2 > *, T2 >>>::iterator _positionInNonActives, const typename Poly::PolyType *_expression, ModuleInput::iterator _defaultBoundPosition, bool _isInteger, size_t _id)
- virtual ~Variable ()
- const Value< T1 > & assignment () const
- Value< T1 > & rAssignment ()
- void storeAssignment ()
- const Value< T1 > & lastConsistentAssignment () const
- void setBasic (bool _basic)
- bool isBasic () const
- bool isOriginal () const
- bool is_integer () const
- bool hasBound () const
- bool involvesEquation () const
- EntryID startEntry () const
- EntryID & rStartEntry ()
- size_t size () const
- size_t & rSize ()
- double conflictActivity () const
- void setSupremum (const Bound< T1, T2 > *_supremum)
- const Bound< T1, T2 > * pSupremum () const
- const Bound< T1, T2 > & supremum () const
- void setInfimum (const Bound< T1, T2 > *_infimum)
- const Bound< T1, T2 > * pInfimum () const
- const Bound< T1, T2 > & infimum () const
- size_t position () const
- size_t id () const
- void setPosition (size_t _position)
- bool isConflicting () const
- std::list< std::list< std::pair< Variable< T1, T2 > *, T2 > > >::iterator positionInNonActives () const
- void setPositionInNonActives (typename std::list< std::list< std::pair< Variable< T1, T2 > *, T2 >>>::iterator _positionInNonActives)
- size_t rLowerBoundsSize ()
- size_t rUpperBoundsSize ()
- const Bound< T1, T2 >::BoundSet & upperbounds () const
- const Bound< T1, T2 >::BoundSet & lowerbounds () const
- Bound< T1, T2 >::BoundSet & rUpperbounds ()
- Bound< T1, T2 >::BoundSet & rLowerbounds ()
- size_t & rPosition ()
- const Poly::PolyType * pExpression () const
- const Poly::PolyType & expression () const
- const T2 & factor () const
- T2 & rFactor ()
- unsigned isSatisfiedBy (const RationalAssignment &_ass) const
- FormulaT inConflictWith (const RationalAssignment &_ass) const
- void updateConflictActivity ()
- std::pair< const Bound< T1, T2 > *, bool > addUpperBound (Value< T1 > *const _val, ModuleInput::iterator _position, const FormulaT &_constraint, bool _deduced=false)
- std::pair< const Bound< T1, T2 > *, bool > addLowerBound (Value< T1 > *const _val, ModuleInput::iterator _position, const FormulaT &_constraint, bool _deduced=false)
- std::pair< const Bound< T1, T2 > *, bool > addEqualBound (Value< T1 > *const _val, ModuleInput::iterator _position, const FormulaT &_constraint)

- void removeBound (const Bound< T1, T2 > *_bound)
- bool deactivateBound (const Bound< T1, T2 > *bound, ModuleInput::iterator _position)
- RationalInterval getVariableBounds () const
- FormulasT getDefiningOrigins () const
- bool operator< (const Variable &_variable) const
- bool operator> (const Variable &_variable) const
- bool operator== (const Variable &_variable) const
- bool operator!= (const Variable &_variable) const
- void print (std::ostream &_out=std::cout) const
- void printAllBounds (std::ostream &_out=std::cout, const std::string _init="") const

0.15.551.1 Constructor & Destructor Documentation

0.15.551.1.1 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	

0.15.551.1.2 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	

0.15.551.1.3 ~Variable()

```
template<typename T1 , typename T2 >
virtual smtrat::lra::Variable< T1, T2 >::~Variable ( ) [virtual]
```

0.15.551.2 Member Function Documentation

0.15.551.2.1 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

0.15.551.2.2 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

0.15.551.2.3 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

0.15.551.2.4 assignment()

```
template<typename T1 , typename T2 >
const Value<T1>& smtrat::lra::Variable< T1, T2 >::assignment ( ) const  [inline]
```

Returns

0.15.551.2.5 conflictActivity()

```
template<typename T1 , typename T2 >
double smtrat::lra::Variable< T1, T2 >::conflictActivity ( ) const  [inline]
```

Returns

0.15.551.2.6 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

0.15.551.2.7 expression()

```
template<typename T1 , typename T2 >
const Poly::PolyType& smtrat::lra::Variable< T1, T2 >::expression ( ) const  [inline]
```

Returns

0.15.551.2.8 factor()

```
template<typename T1 , typename T2 >
const T2& smtrat::lra::Variable< T1, T2 >::factor ( ) const  [inline]
```

Returns

0.15.551.2.9 getDefiningOrigins()

```
template<typename T1 , typename T2 >
FormulasT smtrat::lra::Variable< T1, T2 >::getDefiningOrigins ( ) const
```

Returns

0.15.551.2.10 getVariableBounds()

```
template<typename T1 , typename T2 >
RationalInterval smtrat::lra::Variable< T1, T2 >::getVariableBounds ( ) const
```

Returns

0.15.551.2.11 hasBound()

```
template<typename T1 , typename T2 >
bool smtrat::lra::Variable< T1, T2 >::hasBound ( ) const  [inline]
```

Returns

0.15.551.2.12 id()

```
template<typename T1 , typename T2 >
size_t smtrat::lra::Variable< T1, T2 >::id ( ) const  [inline]
```

0.15.551.2.13 inConflictWith()

```
template<typename T1 , typename T2 >
FormulaT smtrat::lra::Variable< T1, T2 >::inConflictWith (
            const RationalAssignment & _ass ) const  [inline]
```

Parameters

_ass	

Returns

0.15.551.2.14 infimum()

```
template<typename T1 , typename T2 >
const Bound<T1, T2>& smtrat::lra::Variable< T1, T2 >::infimum ( ) const  [inline]
```

Returns

0.15.551.2.15 involvesEquation()

```
template<typename T1 , typename T2 >
bool smtrat::lra::Variable< T1, T2 >::involvesEquation ( ) const  [inline]
```

Returns

0.15.551.2.16 is_integer()

```
template<typename T1 , typename T2 >
bool smtrat::lra::Variable< T1, T2 >::is_integer ( ) const  [inline]
```

Returns

0.15.551.2.17 isBasic()

```
template<typename T1 , typename T2 >
bool smtrat::lra::Variable< T1, T2 >::isBasic ( ) const [inline]
```

Returns

0.15.551.2.18 isConflicting()

```
template<typename T1 , typename T2 >
bool smtrat::lra::Variable< T1, T2 >::isConflicting ( ) const [inline]
```

0.15.551.2.19 isOriginal()

```
template<typename T1 , typename T2 >
bool smtrat::lra::Variable< T1, T2 >::isOriginal ( ) const [inline]
```

Returns

0.15.551.2.20 isSatisfiedBy()

```
template<typename T1 , typename T2 >
unsigned smtrat::lra::Variable< T1, T2 >::isSatisfiedBy (
            const RationalAssignment & _ass ) const [inline]
```

Parameters

_ass	

Returns

0.15.551.2.21 lastConsistentAssignment()

```
template<typename T1 , typename T2 >
const Value<T1>& smtrat::lra::Variable< T1, T2 >::lastConsistentAssignment ( ) const [inline]
```

Returns

0.15.551.2.22 lowerbounds()

```
template<typename T1 , typename T2 >
const Bound<T1, T2>::BoundSet& smtrat::lra::Variable< T1, T2 >::lowerbounds ( ) const [inline]
```

Returns

0.15.551.2.23 operator"!=()

```
template<typename T1 , typename T2 >
bool smtrat::lra::Variable< T1, T2 >::operator!= (
            const Variable< T1, T2 > & _variable ) const [inline]
```

0.15.551.2.24 operator<()

```
template<typename T1 , typename T2 >
bool smtrat::lra::Variable< T1, T2 >::operator< (
            const Variable< T1, T2 > & _variable ) const [inline]
```

Parameters

_bound	

Returns

0.15.551.2.25 operator==()

```
template<typename T1 , typename T2 >
bool smtrat::lra::Variable< T1, T2 >::operator== (
            const Variable< T1, T2 > & _variable ) const  [inline]
```

0.15.551.2.26 operator>()

```
template<typename T1 , typename T2 >
bool smtrat::lra::Variable< T1, T2 >::operator> (
            const Variable< T1, T2 > & _variable ) const  [inline]
```

0.15.551.2.27 pExpression()

```
template<typename T1 , typename T2 >
const Poly::PolyType* smtrat::lra::Variable< T1, T2 >::pExpression ( ) const  [inline]
```

Returns

0.15.551.2.28 pInfimum()

```
template<typename T1 , typename T2 >
const Bound<T1, T2>* smtrat::lra::Variable< T1, T2 >::pInfimum ( ) const  [inline]
```

Returns

0.15.551.2.29 position()

```
template<typename T1 , typename T2 >
size_t smtrat::lra::Variable< T1, T2 >::position ( ) const  [inline]
```

Returns

0.15.551.2.30 positionInNonActives()

```
template<typename T1 , typename T2 >
std::list<std::list<std::pair<Variable<T1,T2>*,T2> > >::iterator smtrat::lra::Variable< T1,
T2 >::positionInNonActives ( ) const  [inline]
```

Returns

0.15.551.2.31 print()

```
template<typename T1 , typename T2 >
void smtrat::lra::Variable< T1, T2 >::print (
            std::ostream & _out = std::cout ) const
```

Parameters

_out	

0.15.551.2.32 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	

0.15.551.2.33 pSupremum()

```
template<typename T1 , typename T2 >
const Bound<T1, T2>* smtrat::lra::Variable< T1, T2 >::pSupremum ( ) const  [inline]
```

Returns

0.15.551.2.34 rAssignment()

```
template<typename T1 , typename T2 >
Value<T1>& smtrat::lra::Variable< T1, T2 >::rAssignment ( )  [inline]
```

Returns

0.15.551.2.35 removeBound()

```
template<typename T1 , typename T2 >
void smtrat::lra::Variable< T1, T2 >::removeBound (
            const Bound< T1, T2 > * _bound )
```

Parameters

_bound	

0.15.551.2.36 rFactor()

```
template<typename T1 , typename T2 >
T2& smtrat::lra::Variable< T1, T2 >::rFactor ( )  [inline]
```

Returns

0.15.551.2.37 rLowerbounds()

```
template<typename T1 , typename T2 >
Bound<T1, T2>::BoundSet& smtrat::lra::Variable< T1, T2 >::rLowerbounds ( )  [inline]
```

Returns

0.15.551.2.38 rLowerBoundsSize()

`template<typename T1 , typename T2 >`
`size_t smtrat::lra::Variable< T1, T2 >::rLowerBoundsSize ( ) [inline]`

Returns

0.15.551.2.39 rPosition()

`template<typename T1 , typename T2 >`
`size_t& smtrat::lra::Variable< T1, T2 >::rPosition ( ) [inline]`

Returns

0.15.551.2.40 rSize()

`template<typename T1 , typename T2 >`
`size_t& smtrat::lra::Variable< T1, T2 >::rSize ( ) [inline]`

Returns

0.15.551.2.41 rStartEntry()

`template<typename T1 , typename T2 >`
`EntryID& smtrat::lra::Variable< T1, T2 >::rStartEntry ( ) [inline]`

Returns

0.15.551.2.42 rUpperbounds()

`template<typename T1 , typename T2 >`
`Bound<T1, T2>::BoundSet& smtrat::lra::Variable< T1, T2 >::rUpperbounds ( ) [inline]`

Returns

0.15.551.2.43 rUpperBoundsSize()

`template<typename T1 , typename T2 >`
`size_t smtrat::lra::Variable< T1, T2 >::rUpperBoundsSize ( ) [inline]`

Returns

0.15.551.2.44 setBasic()

`template<typename T1 , typename T2 >`
`void smtrat::lra::Variable< T1, T2 >::setBasic (`
`bool _basic ) [inline]`

Parameters

_basic	

0.15.551.2.45 setInfimum()

```
template<typename T1 , typename T2 >
void smtrat::lra::Variable< T1, T2 >::setInfimum (
            const Bound< T1, T2 > * _infimum )  [inline]
```

Parameters

_infimum	

0.15.551.2.46 setPosition()

```
template<typename T1 , typename T2 >
void smtrat::lra::Variable< T1, T2 >::setPosition (
            size_t _position )  [inline]
```

Parameters

_position	

0.15.551.2.47 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	

0.15.551.2.48 setSupremum()

```
template<typename T1 , typename T2 >
void smtrat::lra::Variable< T1, T2 >::setSupremum (
            const Bound< T1, T2 > * _supremum )  [inline]
```

Parameters

_supremum	

0.15.551.2.49 size()

```
template<typename T1 , typename T2 >
size_t smtrat::lra::Variable< T1, T2 >::size ( ) const  [inline]
```

Returns

0.15.551.2.50 startEntry()

```
template<typename T1 , typename T2 >
EntryID smtrat::lra::Variable< T1, T2 >::startEntry ( ) const  [inline]
```

Returns

0.15.551.2.51 storeAssignment()

`template<typename T1 , typename T2 >`
`void smtrat::lra::Variable< T1, T2 >::storeAssignment ( ) [inline]`

0.15.551.2.52 supremum()

`template<typename T1 , typename T2 >`
`const Bound<T1, T2>& smtrat::lra::Variable< T1, T2 >::supremum ( ) const [inline]`

Returns

0.15.551.2.53 updateConflictActivity()

`template<typename T1 , typename T2 >`
`void smtrat::lra::Variable< T1, T2 >::updateConflictActivity ( ) [inline]`

0.15.551.2.54 upperbounds()

`template<typename T1 , typename T2 >`
`const Bound<T1, T2>::BoundSet& smtrat::lra::Variable< T1, T2 >::upperbounds ( ) const [inline]`

Returns

0.15.552 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

0.15.552.1 Detailed Description

template<typename T>
class smtrat::vb::Variable< T >

Class for a variable.

0.15.552.2 Constructor & Destructor Documentation

0.15.552.2.1 Variable() `template<typename T >`

`smtrat::vb::Variable< T >::Variable ( )`

Constructs this variable.

0.15.552.2.2 ~Variable() `template<typename T >`

`smtrat::vb::Variable< T >::~Variable ( )`

0.15.552.3 Member Function Documentation

0.15.552.3.1 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.

0.15.552.3.2 conflicting() `template<typename T >`

`bool smtrat::vb::Variable< T >::conflicting ( ) const`

Returns

true, if there is a conflict; false, otherwise.

0.15.552.3.3 doubleIntervalHasBeenUpdated() `template<typename T >`

`void smtrat::vb::Variable< T >::doubleIntervalHasBeenUpdated ( ) const [inline]`

Sets the flag indicating that the stored double interval representing the variable's bounds is up to date to false.

0.15.552.3.4 exactIntervalHasBeenUpdated() `template<typename T >`

`void smtrat::vb::Variable< T >::exactIntervalHasBeenUpdated ( ) const [inline]`

Sets the flag indicating that the stored exact interval representing the variable's bounds is up to date to false.

0.15.552.3.5 infimum() `template<typename T >`

`const Bound<T>& smtrat::vb::Variable< T >::infimum ( ) const [inline]`

Returns

A constant reference to the infimum of this variable.

0.15.552.3.6 lowerbounds() `template<typename T >`

`const BoundSet& smtrat::vb::Variable< T >::lowerbounds ( ) const [inline]`

Returns

A constant reference to the set of lower bounds of this variable.

0.15.552.3.7 pInfimum() `template<typename T >`

`const Bound<T>* smtrat::vb::Variable< T >::pInfimum ( ) const [inline]`

Returns

A pointer to the infimum of this variable.

0.15.552.3.8 pSupremum() `template<typename T >`

`const Bound<T>* smtrat::vb::Variable< T >::pSupremum ( ) const [inline]`

Returns

A pointer to the supremum of this variable.

0.15.552.3.9 supremum() `template<typename T >`

`const Bound<T>& smtrat::vb::Variable< T >::supremum ( ) const [inline]`

Returns

A constant reference to the supremum of this variable.

0.15.552.3.10 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.

0.15.552.3.11 updatedDoubleInterval() `template<typename T >`

`bool smtrat::vb::Variable< T >::updatedDoubleInterval ( ) const [inline]`

Returns

true, if the stored double interval representing the variable's bounds is up to date. false, otherwise

0.15.552.3.12 updatedExactInterval() `template<typename T >`

`bool smtrat::vb::Variable< T >::updatedExactInterval ( ) const [inline]`

Returns

true, if the stored exact interval representing the variable's bounds is up to date. false, otherwise

0.15.552.3.13 upperbounds() `template<typename T >`

`const BoundSet& smtrat::vb::Variable< T >::upperbounds ( ) const [inline]`

Returns

A constant reference to the set of upper bounds of this variable.

0.15.553 smtrat::vb::VariableBounds< T > Class Template Reference

Class to manage the bounds of a variable.

`#include <VariableBounds.h>`

Public Types

- typedef std::map< ConstraintT, const Bound< T > * > ConstraintBoundMap

  A map from Constraint pointers to Bound pointers.
- typedef carl::FastMap< carl::Variable, Variable< T > * > VariableMap

  A hash-map from arithmetic variables to variables managing the bounds.

Public Member Functions

- VariableBounds ()

  Constructs a variable bounds manager.
- ~VariableBounds ()

  Destructs a variable bounds manager.
- void clear ()
- bool empty () const
- bool addBound (const ConstraintT &_constraint, const T &_origin)

  Updates the variable bounds by the given constraint.
- bool addBound (const FormulaT &_formula, const T &_origin)
- unsigned removeBound (const ConstraintT &_constraint, const T &_origin)

  Removes all effects the given constraint has on the variable bounds.
- unsigned removeBound (const FormulaT &_formula, const T &_origin)
- unsigned removeBound (const ConstraintT &_constraint, const T &_origin, carl::Variable *&_changed↩ Variable)

  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

0.15.553.1 Detailed Description

template<typename T>
class smtrat::vb::VariableBounds< T >

Class to manage the bounds of a variable.

0.15.553.2 Member Typedef Documentation

0.15.553.2.1 ConstraintBoundMap `template<typename T >`

`typedef std::map<ConstraintT,const Bound<T>*> smtrat::vb::VariableBounds< T >::ConstraintBoundMap`

A map from Constraint pointers to Bound pointers.

0.15.553.2.2 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.

0.15.553.3 Constructor & Destructor Documentation

0.15.553.3.1 VariableBounds() `template<typename T >`

`smtrat::vb::VariableBounds< T >::VariableBounds ( )`

Constructs a variable bounds manager.

0.15.553.3.2 ~VariableBounds() template<typename T >

smtrat::vb::VariableBounds< T >::~VariableBounds ( )

Destructs a variable bounds manager.

0.15.553.4 Member Function Documentation

0.15.553.4.1 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.

0.15.553.4.2 addBound() [2/2] template<typename T >

```
bool smtrat::vb::VariableBounds< T >::addBound (
            const FormulaT & _formula,
            const T & _origin )
```

0.15.553.4.3 clear() template<typename T >

void smtrat::vb::VariableBounds< T >::clear ( )

0.15.553.4.4 empty() template<typename T >

bool smtrat::vb::VariableBounds< T >::empty ( ) const [inline]

0.15.553.4.5 getBoundDeductions() template<typename T >

std::vector<std::pair<std::vector<ConstraintT>, ConstraintT> > smtrat::vb::VariableBounds< T >::getBoundDeductions ( ) const

Returns

The deductions which this variable bounds manager has detected.

0.15.553.4.6 getConflict() template<typename T >

std::set<T> smtrat::vb::VariableBounds< T >::getConflict ( ) const [inline]

Returns

The origins which cause the conflict. This method can only be called, if there is a conflict.

0.15.553.4.7 getDoubleInterval()

```
template<typename T >
const carl::Interval<double>& smtrat::vb::VariableBounds< T >::getDoubleInterval (
            const carl::Variable & _var ) const
```

Creates a double interval corresponding to the variable bounds of the given variable.

Parameters

_var	The variable to compute the variable bounds as double interval for.

Returns

The variable bounds as a double interval.

0.15.553.4.8 getEvalIntervalMap()

```
template<typename T >
const EvalRationalIntervalMap& smtrat::vb::VariableBounds< T >::getEvalIntervalMap ( ) const
```

Creates an evalintervalmap corresponding to the variable bounds.

Returns

The variable bounds as an evalintervalmap.

0.15.553.4.9 getInterval() [1/3]

```
template<typename T >
RationalInterval smtrat::vb::VariableBounds< T >::getInterval (
            carl::Monomial::Arg _mon ) const
```

Creates an interval corresponding to the bounds of the given monomial.

Parameters

_mon	The monomial to compute the bounds as interval for.

Returns

The monomial bounds as an interval.

0.15.553.4.10 getInterval() [2/3]

```
template<typename T >
const RationalInterval& smtrat::vb::VariableBounds< T >::getInterval (
            const carl::Variable & _var ) const
```

Creates an interval corresponding to the variable bounds of the given variable.

Parameters

_var	The variable to compute the variable bounds as interval for.

Returns

The variable bounds as an interval.

0.15.553.4.11 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.

0.15.553.4.12 getIntervalMap() template<typename T >

```
const smtrat::EvalDoubleIntervalMap& smtrat::vb::VariableBounds< T >::getIntervalMap ( ) const
```

Creates an interval map corresponding to the variable bounds.

Returns

The variable bounds as an interval map.

0.15.553.4.13 getOriginSetOfBounds() [1/3] template<typename T >

```
std::set<T> smtrat::vb::VariableBounds< T >::getOriginSetOfBounds ( ) const
```

0.15.553.4.14 getOriginSetOfBounds() [2/3] template<typename T >

```
std::set<T> smtrat::vb::VariableBounds< T >::getOriginSetOfBounds (
            const carl::Variable & _var ) const
```

0.15.553.4.15 getOriginSetOfBounds() [3/3] template<typename T >

```
std::set<T> smtrat::vb::VariableBounds< T >::getOriginSetOfBounds (
            const carl::Variables & _variables ) const
```

0.15.553.4.16 getOriginsOfBounds() [1/3] template<typename T >

```
std::vector<T> smtrat::vb::VariableBounds< T >::getOriginsOfBounds ( ) const
```

Collect the origins to the supremums and infimums of all variables.

Returns

A set of origins corresponding to the supremums and infimums of all variables.

0.15.553.4.17 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.

0.15.553.4.18 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.

0.15.553.4.19 isConflicting() `template<typename T >`

```
bool smtrat::vb::VariableBounds< T >::isConflicting ( ) const  [inline]
```

Returns

true, if there is a conflicting variable; false, otherwise.

0.15.553.4.20 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).

0.15.553.4.21 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.

0.15.553.4.22 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.

0.15.553.4.23 removeBound() [3/3]

```
template<typename T >
unsigned smtrat::vb::VariableBounds< T >::removeBound (
            const FormulaT & _formula,
            const T & _origin )
```

0.15.554 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

0.15.554.1 Constructor & Destructor Documentation

0.15.554.1.1 VariableCapsule()

```
smtrat::VariableCapsule::VariableCapsule (
            const carl::Variable & xVariable,
            const carl::Variable & yVariable,
            const carl::Variable & zVariable )
```

0.15.554.2 Member Function Documentation

0.15.554.2.1 getXVariable()

```
const carl::Variable & smtrat::VariableCapsule::getXVariable ( )
const
```

0.15.554.2.2 getYVariable()

```
const carl::Variable & smtrat::VariableCapsule::getYVariable ( )
const
```

0.15.554.2.3 getZVariable() `const carl::Variable & smtrat::VariableCapsule::getZVariable ( ) const`

0.15.554.3 Field Documentation

0.15.554.3.1 xVariable `carl::Variable smtrat::VariableCapsule::xVariable`

0.15.554.3.2 yVariable `carl::Variable smtrat::VariableCapsule::yVariable`

0.15.554.3.3 zVariable `carl::Variable smtrat::VariableCapsule::zVariable`

0.15.555 smtrat::mcsat::variableordering::VariableIDs Struct Reference

`#include <helper.h>`

Public Member Functions

- std::size_t operator[ ] (carl::Variable v)
- std::size_t operator[ ] (carl::Variable v) const

Data Fields

- std::map< carl::Variable, std::size_t > mIDs

0.15.555.1 Member Function Documentation

0.15.555.1.1 operator[]() [1/2] `std::size_t smtrat::mcsat::variableordering::VariableIDs::operator[] ( carl::Variable v ) [inline]`

0.15.555.1.2 operator[]() [2/2] `std::size_t smtrat::mcsat::variableordering::VariableIDs::operator[] ( carl::Variable v ) const [inline]`

0.15.555.2 Field Documentation

0.15.555.2.1 mIDs `std::map<carl::Variable, std::size_t> smtrat::mcsat::variableordering::↩ VariableIDs::mIDs`

0.15.556 smtrat::VariableRewriteRule Class Reference

`#include <VariableRewriteRule.h>`

Public Member Functions

- VariableRewriteRule (unsigned varNr, const TermT &term, const carl::BitVector &reasons)
- ~VariableRewriteRule ()

Protected Attributes

- unsigned mVarNr

  Rewrite a variable (identified by this number)
- TermT mTerm

  Rewrite with this term.
- carl::BitVector mReasons

  Based on this origins.

0.15.556.1 Constructor & Destructor Documentation

0.15.556.1.1 VariableRewriteRule()

```
smtrat::VariableRewriteRule::VariableRewriteRule (
    unsigned varNr,
    const TermT & term,
    const carl::BitVector & reasons ) [inline]
```

0.15.556.1.2 ~VariableRewriteRule()

```
smtrat::VariableRewriteRule::~VariableRewriteRule ( ) [inline]
```

0.15.556.2 Field Documentation

0.15.556.2.1 mReasons

```
carl::BitVector smtrat::VariableRewriteRule::mReasons [protected]
```

Based on this origins.

0.15.556.2.2 mTerm

```
TermT smtrat::VariableRewriteRule::mTerm [protected]
```

Rewrite with this term.

0.15.556.2.3 mVarNr

```
unsigned smtrat::VariableRewriteRule::mVarNr [protected]
```

Rewrite a variable (identified by this number)

0.15.557 smtrat::parser::conversion::VariantConverter< Res > Struct Template Reference

Converts variants to some type using the Converter class.

`#include <Conversions.h>`

Public Types

- typedef bool result_type

Public Member Functions

- template<typename T >
  bool operator() (const T &t)
- template<BOOST_VARIANT_ENUM_PARAMS(typename T) >
  bool operator() (const boost::variant< BOOST_VARIANT_ENUM_PARAMS(T)> &t)
- template<BOOST_VARIANT_ENUM_PARAMS(typename T) >
  bool operator() (const boost::variant< BOOST_VARIANT_ENUM_PARAMS(T)> &t, Res &r)
- template<typename T >
  Res convert (const T &t)

Data Fields

- Res result
- Converter< Res > converter

0.15.557.1 Detailed Description

template<typename Res>
struct smtrat::parser::conversion::VariantConverter< Res >

Converts variants to some type using the Converter class.

0.15.557.2 Member Typedef Documentation

0.15.557.2.1 result_type template<typename Res >

```
typedef bool smtrat::parser::conversion::VariantConverter< Res >::result_type
```

0.15.557.3 Member Function Documentation

0.15.557.3.1 convert() template<typename Res >

```
template<typename T >
Res smtrat::parser::conversion::VariantConverter< Res >::convert (
            const T & t )  [inline]
```

0.15.557.3.2 operator()() [1/3] template<typename Res >

```
template<BOOST_VARIANT_ENUM_PARAMS(typename T) >
bool smtrat::parser::conversion::VariantConverter< Res >::operator() (
            const boost::variant< BOOST_VARIANT_ENUM_PARAMS(T)> & t )  [inline]
```

0.15.557.3.3 operator()() [2/3] template<typename Res >

```
template<BOOST_VARIANT_ENUM_PARAMS(typename T) >
bool smtrat::parser::conversion::VariantConverter< Res >::operator() (
            const boost::variant< BOOST_VARIANT_ENUM_PARAMS(T)> & t,
            Res & r )  [inline]
```

0.15.557.3.4 operator()() [3/3] template<typename Res >

```
template<typename T >
bool smtrat::parser::conversion::VariantConverter< Res >::operator() (
            const T & t )  [inline]
```

0.15.557.4 Field Documentation

0.15.557.4.1 converter template<typename Res >

```
Converter<Res> smtrat::parser::conversion::VariantConverter< Res >::converter
```

0.15.557.4.2 result template<typename Res >

```
Res smtrat::parser::conversion::VariantConverter< Res >::result
```

0.15.558 smtrat::VariantMap< Key, Value > Class Template Reference

This class is a specialization of `std::map` where the keys are of arbitrary type and the values are of type `boost::variant`.
`#include <VariantMap.h>`

Public Member Functions

- template<typename T , typename Output >
  void assertType (const Key &key, Output out) const

  Asserts that the value that is associated with the given key has a specified type.
- template<typename T >
  bool has (const Key &key) const

  Checks if there is a value of the specified type for the given key.
- template<typename T >
  const T & get (const Key &key) const

  Returns the value associated with the given key as type `T`.

Data Fields

- K **keys**

  STL member.
- T **elements**

  STL member.

0.15.558.1 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.

0.15.558.2 Member Function Documentation

0.15.558.2.1 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.

0.15.558.2.2 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`.

0.15.558.2.3 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.

0.15.558.3 Field Documentation

0.15.558.3.1 elements `T std::map< K, T >::elements [inherited]`

STL member.

0.15.558.3.2 keys `K std::map< K, T >::keys [inherited]`

STL member.

0.15.559 smtrat::parser::conversion::VariantVariantConverter< Res > Struct Template Reference

Converts variants to another variant type not using the Converter class.

`#include <Conversions.h>`

Public Types

- typedef Res result_type

Public Member Functions

- template<typename T >
  Res operator() (const T &t)
- template<typename Variant >
  Res convert (const Variant &t)

0.15.559.1 Detailed Description

template<typename Res>
struct smtrat::parser::conversion::VariantVariantConverter< Res >

Converts variants to another variant type not using the Converter class.

0.15.559.2 Member Typedef Documentation

0.15.559.2.1 result_type `template<typename Res >`

```
typedef Res smtrat::parser::conversion::VariantVariantConverter< Res >::result_type
```

0.15.559.3 Member Function Documentation

0.15.559.3.1 convert() `template<typename Res >`

```
template<typename Variant >
Res smtrat::parser::conversion::VariantVariantConverter< Res >::convert (
            const Variant & t ) [inline]
```

0.15.559.3.2 operator()()

```
template<typename Res >
template<typename T >
Res smtrat::parser::conversion::VariantVariantConverter< Res >::operator() (
            const T & t ) [inline]
```

0.15.560 smtrat::VarSchedulerBase Class Reference

Base class for variable schedulers.

`#include <VarScheduler.h>`

Public Member Functions

- template<typename BaseModule >
  VarSchedulerBase (BaseModule &baseModule)
- void rebuild ()

  Rebuild heap.
- void insert (Minisat::Var)

  Insert a variable.
- Minisat::Lit pop ()

  Returns the next variable to be decided.
- bool empty ()

  Check if empty.
- void print () const

  Check if variable is contained.
- void increaseActivity (Minisat::Var)
- void decreaseActivity (Minisat::Var)
- void rebuildActivities ()
- template<typename Constraints >
  void rebuildTheoryVars (const Constraints &)
- void attachClause (Minisat::CRef)
- void detachClause (Minisat::CRef)
- void relocateClauses (std::function< void(Minisat::CRef &)>)

Protected Attributes

- std::function< double(Minisat::Var)> getActivity
- std::function< char(Minisat::Var)> getPolarity
- std::function< void(Minisat::Var, bool)> setPolarity
- std::function< bool(Minisat::Var)> isDecisionVar
- std::function< bool(Minisat::Var)> isBoolValueUndef
- std::function< bool(Minisat::Var)> isTheoryAbstraction
- std::function< const FormulaT &(Minisat::Var)> reabstractVariable
- std::function< const FormulaT &(Minisat::Lit)> reabstractLiteral
- std::function< const Minisat::Clause &(Minisat::CRef)> getClause
- std::function< Minisat::lbool(Minisat::Var)> getBoolVarValue
- std::function< Minisat::lbool(Minisat::Lit)> getBoolLitValue
- std::function< unsigned(const FormulaT &)> currentlySatisfiedByBackend
- std::function< Minisat::Var(const FormulaT &)> abstractVariable
- std::function< const Minisat::Lit(const FormulaT &)> abstractLiteral
- std::function< bool(const FormulaT &)> isAbstractedFormula

0.15.560.1 Detailed Description

Base class for variable schedulers.
Removes the need to implement stubs.
During instantiation, the Scheduler is constructed with a const SATModule& as parameter.

0.15.560.2 Constructor & Destructor Documentation

0.15.560.2.1 VarSchedulerBase()

```
template<typename BaseModule >
smtrat::VarSchedulerBase::VarSchedulerBase (
            BaseModule & baseModule ) [inline]
```

0.15.560.3 Member Function Documentation

0.15.560.3.1 attachClause()

```
void smtrat::VarSchedulerBase::attachClause (
            Minisat::CRef ) [inline]
```

0.15.560.3.2 decreaseActivity()

```
void smtrat::VarSchedulerBase::decreaseActivity (
            Minisat::Var ) [inline]
```

0.15.560.3.3 detachClause()

```
void smtrat::VarSchedulerBase::detachClause (
            Minisat::CRef ) [inline]
```

0.15.560.3.4 empty()

```
bool smtrat::VarSchedulerBase::empty ( ) [inline]
```

Check if empty.

0.15.560.3.5 increaseActivity()

```
void smtrat::VarSchedulerBase::increaseActivity (
            Minisat::Var ) [inline]
```

0.15.560.3.6 insert()

```
void smtrat::VarSchedulerBase::insert (
            Minisat::Var ) [inline]
```

Insert a variable.
If already contained, do nothing.

0.15.560.3.7 pop()

```
Minisat::Lit smtrat::VarSchedulerBase::pop ( ) [inline]
```

Returns the next variable to be decided.
Returns and removes the next variable to be decided.

0.15.560.3.8 print()

```
void smtrat::VarSchedulerBase::print ( ) const [inline]
```

Check if variable is contained.
Print.

0.15.560.3.9 rebuild()

```
void smtrat::VarSchedulerBase::rebuild ( ) [inline]
```

Rebuild heap.

0.15.560.3.10 rebuildActivities()

```
void smtrat::VarSchedulerBase::rebuildActivities ( ) [inline]
```

0.15.560.3.11 rebuildTheoryVars()

```
template<typename Constraints >
void smtrat::VarSchedulerBase::rebuildTheoryVars (
            const Constraints & ) [inline]
```

0.15.560.3.12 relocateClauses()

`void smtrat::VarSchedulerBase::relocateClauses ( std::function< void(Minisat::CRef &)> ) [inline]`

0.15.560.4 Field Documentation

0.15.560.4.1 abstractLiteral

`std::function<const Minisat::Lit(const FormulaT&)> smtrat::VarSchedulerBase::abstractLiteral [protected]`

0.15.560.4.2 abstractVariable

`std::function< Minisat::Var(const FormulaT&)> smtrat::VarSchedulerBase::abstractVariable [protected]`

0.15.560.4.3 currentlySatisfiedByBackend

`std::function<unsigned(const FormulaT&)> smtrat::VarSchedulerBase::currentlySatisfiedByBackend [protected]`

0.15.560.4.4 getActivity

`std::function<double(Minisat::Var)> smtrat::VarSchedulerBase::getActivity [protected]`

0.15.560.4.5 getBoolLitValue

`std::function<Minisat::lbool(Minisat::Lit)> smtrat::VarSchedulerBase::getBoolLitValue [protected]`

0.15.560.4.6 getBoolVarValue

`std::function<Minisat::lbool(Minisat::Var)> smtrat::VarSchedulerBase::getBoolVarValue [protected]`

0.15.560.4.7 getClause

`std::function<const Minisat::Clause&(Minisat::CRef)> smtrat::VarSchedulerBase::getClause [protected]`

0.15.560.4.8 getPolarity

`std::function<char(Minisat::Var)> smtrat::VarSchedulerBase::getPolarity [protected]`

0.15.560.4.9 isAbstractedFormula

`std::function<bool(const FormulaT&)> smtrat::VarSchedulerBase::isAbstractedFormula [protected]`

0.15.560.4.10 isBoolValueUndef

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isBoolValueUndef [protected]`

0.15.560.4.11 isDecisionVar

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isDecisionVar [protected]`

0.15.560.4.12 isTheoryAbstraction

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isTheoryAbstraction [protected]`

0.15.560.4.13 reabstractLiteral `std::function<const FormulaT&(Minisat::Lit)> smtrat::VarScheduler↩Base::reabstractLiteral [protected]`

0.15.560.4.14 reabstractVariable `std::function<const FormulaT&(Minisat::Var)> smtrat::Var↩SchedulerBase::reabstractVariable [protected]`

0.15.560.4.15 setPolarity `std::function<void(Minisat::Var,bool)> smtrat::VarSchedulerBase↩::setPolarity [protected]`

0.15.561 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

0.15.561.1 Constructor & Destructor Documentation

0.15.561.1.1 VarSchedulerFixedRandom()

```
template<typename BaseModule >
smtrat::VarSchedulerFixedRandom::VarSchedulerFixedRandom (
            BaseModule & baseModule ) [inline], [explicit]
```

0.15.561.2 Member Function Documentation

0.15.561.2.1 attachClause()

```
void smtrat::VarSchedulerBase::attachClause (
            Minisat::CRef ) [inline], [inherited]
```

0.15.561.2.2 decreaseActivity()

```
void smtrat::VarSchedulerMinisat::decreaseActivity (
            Minisat::Var var ) [inline], [inherited]
```

0.15.561.2.3 detachClause()

```
void smtrat::VarSchedulerBase::detachClause (
            Minisat::CRef ) [inline], [inherited]
```

0.15.561.2.4 empty()

```
bool smtrat::VarSchedulerMinisat::empty ( ) [inline], [inherited]
```

0.15.561.2.5 increaseActivity()

```
void smtrat::VarSchedulerMinisat::increaseActivity (
            Minisat::Var var ) [inline], [inherited]
```

0.15.561.2.6 insert()

```
void smtrat::VarSchedulerMinisat::insert (
            Minisat::Var var ) [inline], [inherited]
```

0.15.561.2.7 pop()

```
Minisat::Lit smtrat::VarSchedulerMinisat::pop ( ) [inline], [inherited]
```

0.15.561.2.8 print()

```
void smtrat::VarSchedulerMinisat::print ( ) const [inline], [inherited]
```

0.15.561.2.9 rebuild()

```
void smtrat::VarSchedulerMinisat::rebuild ( ) [inline], [inherited]
```

0.15.561.2.10 rebuildActivities()

```
void smtrat::VarSchedulerMinisat::rebuildActivities ( ) [inline],
[inherited]
```

0.15.561.2.11 rebuildTheoryVars()

```
template<typename Constraints >
void smtrat::VarSchedulerBase::rebuildTheoryVars (
            const Constraints & ) [inline], [inherited]
```

0.15.561.2.12 relocateClauses()

```
void smtrat::VarSchedulerBase::relocateClauses (
            std::function< void(Minisat::CRef &)> ) [inline], [inherited]
```

0.15.561.2.13 top() `Minisat::Var smtrat::VarSchedulerMinisat::top ( ) [inline], [inherited]`

0.15.561.2.14 valid() `bool smtrat::VarSchedulerMinisat::valid ( Minisat::Var var ) [inline], [protected], [inherited]`

0.15.561.3 Field Documentation

0.15.561.3.1 abstractLiteral `std::function<const Minisat::Lit(const FormulaT&)> smtrat::VarSchedulerBase::abstractLiteral [protected], [inherited]`

0.15.561.3.2 abstractVariable `std::function< Minisat::Var(const FormulaT&)> smtrat::VarSchedulerBase::abstractVariable [protected], [inherited]`

0.15.561.3.3 currentlySatisfiedByBackend `std::function<unsigned(const FormulaT&)> smtrat::VarSchedulerBase::currentlySatisfiedByBackend [protected], [inherited]`

0.15.561.3.4 getActivity `std::function<double(Minisat::Var)> smtrat::VarSchedulerBase::getActivity [protected], [inherited]`

0.15.561.3.5 getBoolLitValue `std::function<Minisat::lbool(Minisat::Lit)> smtrat::VarSchedulerBase::getBoolLitValue [protected], [inherited]`

0.15.561.3.6 getBoolVarValue `std::function<Minisat::lbool(Minisat::Var)> smtrat::VarSchedulerBase::getBoolVarValue [protected], [inherited]`

0.15.561.3.7 getClause `std::function<const Minisat::Clause&(Minisat::CRef)> smtrat::VarSchedulerBase::getClause [protected], [inherited]`

0.15.561.3.8 getPolarity `std::function<char(Minisat::Var)> smtrat::VarSchedulerBase::getPolarity [protected], [inherited]`

0.15.561.3.9 isAbstractedFormula `std::function<bool(const FormulaT&)> smtrat::VarSchedulerBase::isAbstractedFormula [protected], [inherited]`

0.15.561.3.10 isBoolValueUndef `std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isBoolValueUndef [protected], [inherited]`

0.15.561.3.11 isDecisionVar `std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isDecisionVar [protected], [inherited]`

0.15.561.3.12 isTheoryAbstraction `std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isTheoryAbstraction [protected], [inherited]`

0.15.561.3.13 reabstractLiteral `std::function<const FormulaT&(Minisat::Lit)> smtrat::VarSchedulerBase::reabstractLiteral [protected], [inherited]`

0.15.561.3.14 reabstractVariable `std::function<const FormulaT&(Minisat::Var)> smtrat::VarSchedulerBase::reabstractVariable [protected], [inherited]`

0.15.561.3.15 setPolarity `std::function<void(Minisat::Var,bool)> smtrat::VarSchedulerBase::setPolarity [protected], [inherited]`

0.15.562 smtrat::VarSchedulerMcsatActivityPreferTheory< vot > Class Template Reference

Activity-based scheduler preferring initially theory variables.
`#include <VarSchedulerMcsat.h>`

Public Member Functions

- template<typename BaseModule >
  VarSchedulerMcsatActivityPreferTheory (BaseModule &baseModule)
- void rebuild ()
- void insert (Minisat::Var var)
- Minisat::Lit pop ()
- bool empty ()
- void print () const
- void increaseActivity (Minisat::Var var)
- void decreaseActivity (Minisat::Var var)
- void rebuildActivities ()
- template<typename Constraints >
  void rebuildTheoryVars (const Constraints &c)
- void attachClause (Minisat::CRef)
- void detachClause (Minisat::CRef)
- void relocateClauses (std::function< void(Minisat::CRef &)>)

Protected Attributes

- std::function< bool(Minisat::Var)> isTheoryVar
- std::function< carl::Variable(Minisat::Var)> carlVar
- std::function< Minisat::Var(carl::Variable)> minisatVar
- std::function< Model()> currentModel
- std::function< double(Minisat::Var)> getActivity
- std::function< char(Minisat::Var)> getPolarity
- std::function< void(Minisat::Var, bool)> setPolarity
- std::function< bool(Minisat::Var)> isDecisionVar
- std::function< bool(Minisat::Var)> isBoolValueUndef
- std::function< bool(Minisat::Var)> isTheoryAbstraction
- std::function< const FormulaT &(Minisat::Var)> reabstractVariable
- std::function< const FormulaT &(Minisat::Lit)> reabstractLiteral
- std::function< const Minisat::Clause &(Minisat::CRef)> getClause
- std::function< Minisat::lbool(Minisat::Var)> getBoolVarValue
- std::function< Minisat::lbool(Minisat::Lit)> getBoolLitValue

- std::function< unsigned(const FormulaT &)> currentlySatisfiedByBackend
- std::function< Minisat::Var(const FormulaT &)> abstractVariable
- std::function< const Minisat::Lit(const FormulaT &)> abstractLiteral
- std::function< bool(const FormulaT &)> isAbstractedFormula

0.15.562.1 Detailed Description

template<mcsat::VariableOrdering vot>
class smtrat::VarSchedulerMcsatActivityPreferTheory< vot >

Activity-based scheduler preferring initially theory variables.

0.15.562.2 Constructor & Destructor Documentation

0.15.562.2.1 VarSchedulerMcsatActivityPreferTheory()

```
template<mcsat::VariableOrdering vot>
template<typename BaseModule >
smtrat::VarSchedulerMcsatActivityPreferTheory< vot >::VarSchedulerMcsatActivityPreferTheory (
            BaseModule & baseModule )  [inline], [explicit]
```

0.15.562.3 Member Function Documentation

0.15.562.3.1 attachClause()

```
void smtrat::VarSchedulerBase::attachClause (
            Minisat::CRef  )  [inline], [inherited]
```

0.15.562.3.2 decreaseActivity()

```
template<mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatActivityPreferTheory< vot >::decreaseActivity (
            Minisat::Var var )  [inline]
```

0.15.562.3.3 detachClause()

```
void smtrat::VarSchedulerBase::detachClause (
            Minisat::CRef  )  [inline], [inherited]
```

0.15.562.3.4 empty()

```
template<mcsat::VariableOrdering vot>
bool smtrat::VarSchedulerMcsatActivityPreferTheory< vot >::empty ( )  [inline]
```

0.15.562.3.5 increaseActivity()

```
template<mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatActivityPreferTheory< vot >::increaseActivity (
            Minisat::Var var )  [inline]
```

0.15.562.3.6 insert()

```
template<mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatActivityPreferTheory< vot >::insert (
            Minisat::Var var )  [inline]
```

0.15.562.3.7 pop()

```
template<mcsat::VariableOrdering vot>
Minisat::Lit smtrat::VarSchedulerMcsatActivityPreferTheory< vot >::pop ( )  [inline]
```

0.15.562.3.8 print()

```
template<mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatActivityPreferTheory< vot >::print ( ) const  [inline]
```

0.15.562.3.9 rebuild()

```
template<mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatActivityPreferTheory< vot >::rebuild ( )  [inline]
```

0.15.562.3.10 rebuildActivities()

```
template<mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatActivityPreferTheory< vot >::rebuildActivities ( )  [inline]
```

0.15.562.3.11 rebuildTheoryVars()

```
template<mcsat::VariableOrdering vot>
template<typename Constraints >
void smtrat::VarSchedulerMcsatActivityPreferTheory< vot >::rebuildTheoryVars (
            const Constraints & c )  [inline]
```

0.15.562.3.12 relocateClauses()

```
void smtrat::VarSchedulerBase::relocateClauses (
            std::function< void(Minisat::CRef &)> )  [inline], [inherited]
```

0.15.562.4 Field Documentation

0.15.562.4.1 abstractLiteral

```
std::function<const Minisat::Lit(const FormulaT&)> smtrat::Var↩
SchedulerBase::abstractLiteral  [protected], [inherited]
```

0.15.562.4.2 abstractVariable

```
std::function< Minisat::Var(const FormulaT&)> smtrat::VarScheduler↩
Base::abstractVariable  [protected], [inherited]
```

0.15.562.4.3 carlVar

```
std::function<carl::Variable(Minisat::Var)> smtrat::VarSchedulerMcsat↩
Base::carlVar  [protected], [inherited]
```

0.15.562.4.4 currentlySatisfiedByBackend

```
std::function<unsigned(const FormulaT&)> smtrat::Var↩
SchedulerBase::currentlySatisfiedByBackend  [protected], [inherited]
```

0.15.562.4.5 currentModel

```
std::function<Model()> smtrat::VarSchedulerMcsatBase::currentModel
[protected], [inherited]
```

0.15.562.4.6 getActivity

```
std::function<double(Minisat::Var)> smtrat::VarSchedulerBase::get↩
Activity  [protected], [inherited]
```

0.15.562.4.7 getBoolLitValue

```
std::function<Minisat::lbool(Minisat::Lit)> smtrat::VarScheduler↩
Base::getBoolLitValue  [protected], [inherited]
```

0.15.562.4.8 getBoolVarValue

```
std::function<Minisat::lbool(Minisat::Var)> smtrat::VarScheduler↩
Base::getBoolVarValue  [protected], [inherited]
```

0.15.562.4.9 getClause

`std::function<const Minisat::Clause&(Minisat::CRef)> smtrat::VarSchedulerBase::getClause [protected], [inherited]`

0.15.562.4.10 getPolarity

`std::function<char(Minisat::Var)> smtrat::VarSchedulerBase::getPolarity [protected], [inherited]`

0.15.562.4.11 isAbstractedFormula

`std::function<bool(const FormulaT&)> smtrat::VarSchedulerBase::isAbstractedFormula [protected], [inherited]`

0.15.562.4.12 isBoolValueUndef

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isBoolValueUndef [protected], [inherited]`

0.15.562.4.13 isDecisionVar

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isDecisionVar [protected], [inherited]`

0.15.562.4.14 isTheoryAbstraction

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isTheoryAbstraction [protected], [inherited]`

0.15.562.4.15 isTheoryVar

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerMcsatBase::isTheoryVar [protected], [inherited]`

0.15.562.4.16 minisatVar

`std::function<Minisat::Var(carl::Variable)> smtrat::VarSchedulerMcsatBase::minisatVar [protected], [inherited]`

0.15.562.4.17 reabstractLiteral

`std::function<const FormulaT&(Minisat::Lit)> smtrat::VarSchedulerBase::reabstractLiteral [protected], [inherited]`

0.15.562.4.18 reabstractVariable

`std::function<const FormulaT&(Minisat::Var)> smtrat::VarSchedulerBase::reabstractVariable [protected], [inherited]`

0.15.562.4.19 setPolarity

`std::function<void(Minisat::Var,bool)> smtrat::VarSchedulerBase::setPolarity [protected], [inherited]`

0.15.563 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

0.15.563.1 Detailed Description

Base class for all MCSAT variable scheduler.
Should not be used directly.

0.15.563.2 Constructor & Destructor Documentation

0.15.563.2.1 VarSchedulerMcsatBase()

```
template<typename BaseModule >
smtrat::VarSchedulerMcsatBase::VarSchedulerMcsatBase (
            BaseModule & baseModule ) [inline]
```

0.15.563.3 Member Function Documentation

0.15.563.3.1 attachClause()

`void smtrat::VarSchedulerBase::attachClause ( Minisat::CRef ) [inline], [inherited]`

0.15.563.3.2 decreaseActivity()

`void smtrat::VarSchedulerBase::decreaseActivity ( Minisat::Var ) [inline], [inherited]`

0.15.563.3.3 detachClause()

`void smtrat::VarSchedulerBase::detachClause ( Minisat::CRef ) [inline], [inherited]`

0.15.563.3.4 empty()

`bool smtrat::VarSchedulerBase::empty ( ) [inline], [inherited]`

Check if empty.

0.15.563.3.5 increaseActivity()

`void smtrat::VarSchedulerBase::increaseActivity ( Minisat::Var ) [inline], [inherited]`

0.15.563.3.6 insert()

`void smtrat::VarSchedulerBase::insert ( Minisat::Var ) [inline], [inherited]`

Insert a variable.
If already contained, do nothing.

0.15.563.3.7 pop()

`Minisat::Lit smtrat::VarSchedulerBase::pop ( ) [inline], [inherited]`

Returns the next variable to be decided.
Returns and removes the next variable to be decided.

0.15.563.3.8 print()

`void smtrat::VarSchedulerBase::print ( ) const [inline], [inherited]`

Check if variable is contained.
Print.

0.15.563.3.9 rebuild()

`void smtrat::VarSchedulerBase::rebuild ( ) [inline], [inherited]`

Rebuild heap.

0.15.563.3.10 rebuildActivities()

`void smtrat::VarSchedulerBase::rebuildActivities ( ) [inline], [inherited]`

0.15.563.3.11 rebuildTheoryVars()

`template<typename Constraints > void smtrat::VarSchedulerBase::rebuildTheoryVars ( const Constraints & ) [inline], [inherited]`

0.15.563.3.12 relocateClauses()

`void smtrat::VarSchedulerBase::relocateClauses ( std::function< void(Minisat::CRef &)> ) [inline], [inherited]`

0.15.563.4 Field Documentation

0.15.563.4.1 abstractLiteral

`std::function<const Minisat::Lit(const FormulaT&)> smtrat::VarSchedulerBase::abstractLiteral [protected], [inherited]`

0.15.563.4.2 abstractVariable

`std::function< Minisat::Var(const FormulaT&)> smtrat::VarScheduler↩ Base::abstractVariable [protected], [inherited]`

0.15.563.4.3 carlVar

`std::function<carl::Variable(Minisat::Var)> smtrat::VarSchedulerMcsat↩ Base::carlVar [protected]`

0.15.563.4.4 currentlySatisfiedByBackend

`std::function<unsigned(const FormulaT&)> smtrat::Var↩ SchedulerBase::currentlySatisfiedByBackend [protected], [inherited]`

0.15.563.4.5 currentModel

`std::function<Model()> smtrat::VarSchedulerMcsatBase::currentModel [protected]`

0.15.563.4.6 getActivity

`std::function<double(Minisat::Var)> smtrat::VarSchedulerBase::get↩ Activity [protected], [inherited]`

0.15.563.4.7 getBoolLitValue

`std::function<Minisat::lbool(Minisat::Lit)> smtrat::VarScheduler↩ Base::getBoolLitValue [protected], [inherited]`

0.15.563.4.8 getBoolVarValue

`std::function<Minisat::lbool(Minisat::Var)> smtrat::VarScheduler↩ Base::getBoolVarValue [protected], [inherited]`

0.15.563.4.9 getClause

`std::function<const Minisat::Clause&(Minisat::CRef)> smtrat::Var↩ SchedulerBase::getClause [protected], [inherited]`

0.15.563.4.10 getPolarity

`std::function<char(Minisat::Var)> smtrat::VarSchedulerBase::get↩ Polarity [protected], [inherited]`

0.15.563.4.11 isAbstractedFormula

`std::function<bool(const FormulaT&)> smtrat::VarScheduler↩ Base::isAbstractedFormula [protected], [inherited]`

0.15.563.4.12 isBoolValueUndef

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase↩ ::isBoolValueUndef [protected], [inherited]`

0.15.563.4.13 isDecisionVar

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::is↩ DecisionVar [protected], [inherited]`

0.15.563.4.14 isTheoryAbstraction

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase↩ ::isTheoryAbstraction [protected], [inherited]`

0.15.563.4.15 isTheoryVar

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerMcsatBase↩ ::isTheoryVar [protected]`

0.15.563.4.16 minisatVar `std::function<Minisat::Var(carl::Variable)> smtrat::VarScheduler↩McsatBase::minisatVar [protected]`

0.15.563.4.17 reabstractLiteral `std::function<const FormulaT&(Minisat::Lit)> smtrat::VarScheduler↩Base::reabstractLiteral [protected], [inherited]`

0.15.563.4.18 reabstractVariable `std::function<const FormulaT&(Minisat::Var)> smtrat::Var↩SchedulerBase::reabstractVariable [protected], [inherited]`

0.15.563.4.19 setPolarity `std::function<void(Minisat::Var,bool)> smtrat::VarSchedulerBase↩::setPolarity [protected], [inherited]`

0.15.564 smtrat::VarSchedulerMcsatBooleanFirst< vot > Class Template Reference

Variable scheduling that all decides Boolean variables first before deciding any theory variable.
`#include <VarSchedulerMcsat.h>`

Public Member Functions

- template<typename BaseModule >
  VarSchedulerMcsatBooleanFirst (BaseModule &baseModule)
- void rebuild ()
- void insert (Minisat::Var var)
- Minisat::Lit pop ()
- bool empty ()
- void print () const
- void increaseActivity (Minisat::Var var)
- void decreaseActivity (Minisat::Var var)
- void rebuildActivities ()
- template<typename Constraints >
  void rebuildTheoryVars (const Constraints &c)
- void attachClause (Minisat::CRef)
- void detachClause (Minisat::CRef)
- void relocateClauses (std::function< void(Minisat::CRef &)>)

Protected Attributes

- std::function< bool(Minisat::Var)> isTheoryVar
- std::function< carl::Variable(Minisat::Var)> carlVar
- std::function< Minisat::Var(carl::Variable)> minisatVar
- std::function< Model()> currentModel
- std::function< double(Minisat::Var)> getActivity
- std::function< char(Minisat::Var)> getPolarity
- std::function< void(Minisat::Var, bool)> setPolarity
- std::function< bool(Minisat::Var)> isDecisionVar
- std::function< bool(Minisat::Var)> isBoolValueUndef
- std::function< bool(Minisat::Var)> isTheoryAbstraction
- std::function< const FormulaT &(Minisat::Var)> reabstractVariable
- std::function< const FormulaT &(Minisat::Lit)> reabstractLiteral
- std::function< const Minisat::Clause &(Minisat::CRef)> getClause
- std::function< Minisat::lbool(Minisat::Var)> getBoolVarValue
- std::function< Minisat::lbool(Minisat::Lit)> getBoolLitValue
- std::function< unsigned(const FormulaT &)> currentlySatisfiedByBackend

- std::function< Minisat::Var(const FormulaT &)> abstractVariable
- std::function< const Minisat::Lit(const FormulaT &)> abstractLiteral
- std::function< bool(const FormulaT &)> isAbstractedFormula

0.15.564.1 Detailed Description

template<mcsat::VariableOrdering vot>
class smtrat::VarSchedulerMcsatBooleanFirst< vot >

Variable scheduling that all decides Boolean variables first before deciding any theory variable.

0.15.564.2 Constructor & Destructor Documentation

0.15.564.2.1 VarSchedulerMcsatBooleanFirst()

```
template<mcsat::VariableOrdering vot>
template<typename BaseModule >
smtrat::VarSchedulerMcsatBooleanFirst< vot >::VarSchedulerMcsatBooleanFirst (
            BaseModule & baseModule )  [inline], [explicit]
```

0.15.564.3 Member Function Documentation

0.15.564.3.1 attachClause()

```
void smtrat::VarSchedulerBase::attachClause (
            Minisat::CRef  )  [inline], [inherited]
```

0.15.564.3.2 decreaseActivity()

```
template<mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatBooleanFirst< vot >::decreaseActivity (
            Minisat::Var var )  [inline]
```

0.15.564.3.3 detachClause()

```
void smtrat::VarSchedulerBase::detachClause (
            Minisat::CRef  )  [inline], [inherited]
```

0.15.564.3.4 empty()

```
template<mcsat::VariableOrdering vot>
bool smtrat::VarSchedulerMcsatBooleanFirst< vot >::empty ( )  [inline]
```

0.15.564.3.5 increaseActivity()

```
template<mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatBooleanFirst< vot >::increaseActivity (
            Minisat::Var var )  [inline]
```

0.15.564.3.6 insert()

```
template<mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatBooleanFirst< vot >::insert (
            Minisat::Var var )  [inline]
```

0.15.564.3.7 pop()

```
template<mcsat::VariableOrdering vot>
Minisat::Lit smtrat::VarSchedulerMcsatBooleanFirst< vot >::pop ( )  [inline]
```

0.15.564.3.8 print()

```
template<mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatBooleanFirst< vot >::print ( ) const [inline]
```

0.15.564.3.9 rebuild()

```
template<mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatBooleanFirst< vot >::rebuild ( ) [inline]
```

0.15.564.3.10 rebuildActivities()

```
template<mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatBooleanFirst< vot >::rebuildActivities ( ) [inline]
```

0.15.564.3.11 rebuildTheoryVars()

```
template<mcsat::VariableOrdering vot>
template<typename Constraints >
void smtrat::VarSchedulerMcsatBooleanFirst< vot >::rebuildTheoryVars (
            const Constraints & c ) [inline]
```

0.15.564.3.12 relocateClauses()

```
void smtrat::VarSchedulerBase::relocateClauses (
            std::function< void(Minisat::CRef &)> ) [inline], [inherited]
```

0.15.564.4 Field Documentation

0.15.564.4.1 abstractLiteral

```
std::function<const Minisat::Lit(const FormulaT&)> smtrat::Var↩
SchedulerBase::abstractLiteral [protected], [inherited]
```

0.15.564.4.2 abstractVariable

```
std::function< Minisat::Var(const FormulaT&)> smtrat::VarScheduler↩
Base::abstractVariable [protected], [inherited]
```

0.15.564.4.3 carlVar

```
std::function<carl::Variable(Minisat::Var)> smtrat::VarSchedulerMcsat↩
Base::carlVar [protected], [inherited]
```

0.15.564.4.4 currentlySatisfiedByBackend

```
std::function<unsigned(const FormulaT&)> smtrat::Var↩
SchedulerBase::currentlySatisfiedByBackend [protected], [inherited]
```

0.15.564.4.5 currentModel

```
std::function<Model()> smtrat::VarSchedulerMcsatBase::currentModel
[protected], [inherited]
```

0.15.564.4.6 getActivity

```
std::function<double(Minisat::Var)> smtrat::VarSchedulerBase::get↩
Activity [protected], [inherited]
```

0.15.564.4.7 getBoolLitValue

```
std::function<Minisat::lbool(Minisat::Lit)> smtrat::VarScheduler↩
Base::getBoolLitValue [protected], [inherited]
```

0.15.564.4.8 getBoolVarValue

```
std::function<Minisat::lbool(Minisat::Var)> smtrat::VarScheduler↩
Base::getBoolVarValue [protected], [inherited]
```

0.15.564.4.9 getClause

`std::function<const Minisat::Clause&(Minisat::CRef)> smtrat::VarSchedulerBase::getClause [protected], [inherited]`

0.15.564.4.10 getPolarity

`std::function<char(Minisat::Var)> smtrat::VarSchedulerBase::getPolarity [protected], [inherited]`

0.15.564.4.11 isAbstractedFormula

`std::function<bool(const FormulaT&)> smtrat::VarSchedulerBase::isAbstractedFormula [protected], [inherited]`

0.15.564.4.12 isBoolValueUndef

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isBoolValueUndef [protected], [inherited]`

0.15.564.4.13 isDecisionVar

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isDecisionVar [protected], [inherited]`

0.15.564.4.14 isTheoryAbstraction

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isTheoryAbstraction [protected], [inherited]`

0.15.564.4.15 isTheoryVar

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerMcsatBase::isTheoryVar [protected], [inherited]`

0.15.564.4.16 minisatVar

`std::function<Minisat::Var(carl::Variable)> smtrat::VarSchedulerMcsatBase::minisatVar [protected], [inherited]`

0.15.564.4.17 reabstractLiteral

`std::function<const FormulaT&(Minisat::Lit)> smtrat::VarSchedulerBase::reabstractLiteral [protected], [inherited]`

0.15.564.4.18 reabstractVariable

`std::function<const FormulaT&(Minisat::Var)> smtrat::VarSchedulerBase::reabstractVariable [protected], [inherited]`

0.15.564.4.19 setPolarity

`std::function<void(Minisat::Var,bool)> smtrat::VarSchedulerBase::setPolarity [protected], [inherited]`

0.15.565 smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler > Class Template Reference

Variable scheduling that all decides theory variables first before deciding any Boolean variable.

`#include <VarSchedulerMcsat.h>`

Public Member Functions

- template<typename BaseModule >
  VarSchedulerMcsatTheoryFirst (BaseModule &baseModule)
- void rebuild ()
- void insert (Minisat::Var var)

- Minisat::Lit pop ()
- bool empty ()
- void print () const
- void increaseActivity (Minisat::Var var)
- void decreaseActivity (Minisat::Var var)
- void rebuildActivities ()
- template<typename Constraints >
  void rebuildTheoryVars (const Constraints &c)
- void attachClause (Minisat::CRef)
- void detachClause (Minisat::CRef)
- void relocateClauses (std::function< void(Minisat::CRef &)>)

Protected Attributes

- std::function< bool(Minisat::Var)> isTheoryVar
- std::function< carl::Variable(Minisat::Var)> carlVar
- std::function< Minisat::Var(carl::Variable)> minisatVar
- std::function< Model()> currentModel
- std::function< double(Minisat::Var)> getActivity
- std::function< char(Minisat::Var)> getPolarity
- std::function< void(Minisat::Var, bool)> setPolarity
- std::function< bool(Minisat::Var)> isDecisionVar
- std::function< bool(Minisat::Var)> isBoolValueUndef
- std::function< bool(Minisat::Var)> isTheoryAbstraction
- std::function< const FormulaT &(Minisat::Var)> reabstractVariable
- std::function< const FormulaT &(Minisat::Lit)> reabstractLiteral
- std::function< const Minisat::Clause &(Minisat::CRef)> getClause
- std::function< Minisat::lbool(Minisat::Var)> getBoolVarValue
- std::function< Minisat::lbool(Minisat::Lit)> getBoolLitValue
- std::function< unsigned(const FormulaT &)> currentlySatisfiedByBackend
- std::function< Minisat::Var(const FormulaT &)> abstractVariable
- std::function< const Minisat::Lit(const FormulaT &)> abstractLiteral
- std::function< bool(const FormulaT &)> isAbstractedFormula

0.15.565.1 Detailed Description

template<typename TheoryScheduler>
class smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >

Variable scheduling that all decides theory variables first before deciding any Boolean variable.

0.15.565.2 Constructor & Destructor Documentation

0.15.565.2.1 VarSchedulerMcsatTheoryFirst()

```
template<typename TheoryScheduler >
template<typename BaseModule >
smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::VarSchedulerMcsatTheoryFirst (
            BaseModule & baseModule )  [inline], [explicit]
```

0.15.565.3 Member Function Documentation

0.15.565.3.1 attachClause()

```
void smtrat::VarSchedulerBase::attachClause (
            Minisat::CRef  )  [inline], [inherited]
```

0.15.565.3.2 decreaseActivity()

```
template<typename TheoryScheduler >
void smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::decreaseActivity (
            Minisat::Var var )  [inline]
```

0.15.565.3.3 detachClause()

```
void smtrat::VarSchedulerBase::detachClause (
            Minisat::CRef  )  [inline], [inherited]
```

0.15.565.3.4 empty()

```
template<typename TheoryScheduler >
bool smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::empty ( )  [inline]
```

0.15.565.3.5 increaseActivity()

```
template<typename TheoryScheduler >
void smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::increaseActivity (
            Minisat::Var var )  [inline]
```

0.15.565.3.6 insert()

```
template<typename TheoryScheduler >
void smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::insert (
            Minisat::Var var )  [inline]
```

0.15.565.3.7 pop()

```
template<typename TheoryScheduler >
Minisat::Lit smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::pop ( )  [inline]
```

0.15.565.3.8 print()

```
template<typename TheoryScheduler >
void smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::print ( ) const  [inline]
```

0.15.565.3.9 rebuild()

```
template<typename TheoryScheduler >
void smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::rebuild ( )  [inline]
```

0.15.565.3.10 rebuildActivities()

```
template<typename TheoryScheduler >
void smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::rebuildActivities ( )  [inline]
```

0.15.565.3.11 rebuildTheoryVars()

```
template<typename TheoryScheduler >
template<typename Constraints >
void smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::rebuildTheoryVars (
            const Constraints & c )  [inline]
```

0.15.565.3.12 relocateClauses()

```
void smtrat::VarSchedulerBase::relocateClauses (
            std::function< void(Minisat::CRef &)>  )  [inline], [inherited]
```

0.15.565.4 Field Documentation

0.15.565.4.1 abstractLiteral

```
std::function<const Minisat::Lit(const FormulaT&)> smtrat::Var↩
SchedulerBase::abstractLiteral  [protected], [inherited]
```

0.15.565.4.2 abstractVariable

`std::function< Minisat::Var(const FormulaT&)> smtrat::VarScheduler↩ Base::abstractVariable [protected], [inherited]`

0.15.565.4.3 carlVar

`std::function<carl::Variable(Minisat::Var)> smtrat::VarSchedulerMcsat↩ Base::carlVar [protected], [inherited]`

0.15.565.4.4 currentlySatisfiedByBackend

`std::function<unsigned(const FormulaT&)> smtrat::Var↩ SchedulerBase::currentlySatisfiedByBackend [protected], [inherited]`

0.15.565.4.5 currentModel

`std::function<Model()> smtrat::VarSchedulerMcsatBase::currentModel [protected], [inherited]`

0.15.565.4.6 getActivity

`std::function<double(Minisat::Var)> smtrat::VarSchedulerBase::get↩ Activity [protected], [inherited]`

0.15.565.4.7 getBoolLitValue

`std::function<Minisat::lbool(Minisat::Lit)> smtrat::VarScheduler↩ Base::getBoolLitValue [protected], [inherited]`

0.15.565.4.8 getBoolVarValue

`std::function<Minisat::lbool(Minisat::Var)> smtrat::VarScheduler↩ Base::getBoolVarValue [protected], [inherited]`

0.15.565.4.9 getClause

`std::function<const Minisat::Clause&(Minisat::CRef)> smtrat::Var↩ SchedulerBase::getClause [protected], [inherited]`

0.15.565.4.10 getPolarity

`std::function<char(Minisat::Var)> smtrat::VarSchedulerBase::get↩ Polarity [protected], [inherited]`

0.15.565.4.11 isAbstractedFormula

`std::function<bool(const FormulaT&)> smtrat::VarScheduler↩ Base::isAbstractedFormula [protected], [inherited]`

0.15.565.4.12 isBoolValueUndef

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase↩ ::isBoolValueUndef [protected], [inherited]`

0.15.565.4.13 isDecisionVar

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::is↩ DecisionVar [protected], [inherited]`

0.15.565.4.14 isTheoryAbstraction

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase↩ ::isTheoryAbstraction [protected], [inherited]`

0.15.565.4.15 isTheoryVar

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerMcsatBase↩ ::isTheoryVar [protected], [inherited]`

0.15.565.4.16 minisatVar

`std::function<Minisat::Var(carl::Variable)> smtrat::VarScheduler↩McsatBase::minisatVar [protected], [inherited]`

0.15.565.4.17 reabstractLiteral

`std::function<const FormulaT&(Minisat::Lit)> smtrat::VarScheduler↩Base::reabstractLiteral [protected], [inherited]`

0.15.565.4.18 reabstractVariable

`std::function<const FormulaT&(Minisat::Var)> smtrat::Var↩SchedulerBase::reabstractVariable [protected], [inherited]`

0.15.565.4.19 setPolarity

`std::function<void(Minisat::Var,bool)> smtrat::VarSchedulerBase↩::setPolarity [protected], [inherited]`

0.15.566 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

0.15.566.1 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.

0.15.566.2 Constructor & Destructor Documentation

0.15.566.2.1 VarSchedulerMcsatUnivariateClausesOnly()

```
template<typename TheoryScheduler , bool respectActivities>
template<typename BaseModule >
smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >::VarSchedulerMcsatUnivar
(
            BaseModule & baseModule )  [inline], [explicit]
```

0.15.566.3 Member Function Documentation

0.15.566.3.1 attachClause()

```
template<typename TheoryScheduler , bool respectActivities>
void smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >↩
::attachClause (
            Minisat::CRef cl )  [inline]
```

0.15.566.3.2 decreaseActivity()

```
template<typename TheoryScheduler , bool respectActivities>
void smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >↩
::decreaseActivity (
            Minisat::Var var )  [inline]
```

0.15.566.3.3 detachClause()

```
template<typename TheoryScheduler , bool respectActivities>
void smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >↩
::detachClause (
            Minisat::CRef cl )  [inline]
```

0.15.566.3.4 empty()

```
template<typename TheoryScheduler , bool respectActivities>
bool smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >↩
::empty ( )  [inline]
```

0.15.566.3.5 increaseActivity()

```
template<typename TheoryScheduler , bool respectActivities>
void smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >↩
::increaseActivity (
            Minisat::Var var )  [inline]
```

0.15.566.3.6 insert()

```
template<typename TheoryScheduler , bool respectActivities>
void smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >↩
::insert (
            Minisat::Var var )  [inline]
```

0.15.566.3.7 pop()

```
template<typename TheoryScheduler , bool respectActivities>
Minisat::Lit smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities
>::pop ( )  [inline]
```

0.15.566.3.8 print()

```
template<typename TheoryScheduler , bool respectActivities>
void smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >↩
::print ( ) const  [inline]
```

0.15.566.3.9 rebuild()

```
template<typename TheoryScheduler , bool respectActivities>
void smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >↩
::rebuild ( )  [inline]
```

0.15.566.3.10 rebuildActivities()

```
template<typename TheoryScheduler , bool respectActivities>
void smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >↩
::rebuildActivities ( )  [inline]
```

0.15.566.3.11 rebuildTheoryVars()

```
template<typename TheoryScheduler , bool respectActivities>
template<typename Constraints >
void smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >↩
::rebuildTheoryVars (
            const Constraints & c )  [inline]
```

0.15.566.3.12 relocateClauses()

```
template<typename TheoryScheduler , bool respectActivities>
void smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >↩
::relocateClauses (
            std::function< void(Minisat::CRef &)> relocate )  [inline]
```

0.15.566.4 Field Documentation

0.15.566.4.1 abstractLiteral

```
std::function<const Minisat::Lit(const FormulaT&)> smtrat::Var↩
SchedulerBase::abstractLiteral  [protected], [inherited]
```

0.15.566.4.2 abstractVariable

```
std::function< Minisat::Var(const FormulaT&)> smtrat::VarScheduler↩
Base::abstractVariable  [protected], [inherited]
```

0.15.566.4.3 carlVar

```
std::function<carl::Variable(Minisat::Var)> smtrat::VarSchedulerMcsat↩
Base::carlVar  [protected], [inherited]
```

0.15.566.4.4 currentlySatisfiedByBackend std::function<unsigned(const FormulaT&)> smtrat::VarSchedulerBase::currentlySatisfiedByBackend [protected], [inherited]

0.15.566.4.5 currentModel std::function<Model()> smtrat::VarSchedulerMcsatBase::currentModel [protected], [inherited]

0.15.566.4.6 getActivity std::function<double(Minisat::Var)> smtrat::VarSchedulerBase::getActivity [protected], [inherited]

0.15.566.4.7 getBoolLitValue std::function<Minisat::lbool(Minisat::Lit)> smtrat::VarSchedulerBase::getBoolLitValue [protected], [inherited]

0.15.566.4.8 getBoolVarValue std::function<Minisat::lbool(Minisat::Var)> smtrat::VarSchedulerBase::getBoolVarValue [protected], [inherited]

0.15.566.4.9 getClause std::function<const Minisat::Clause&(Minisat::CRef)> smtrat::VarSchedulerBase::getClause [protected], [inherited]

0.15.566.4.10 getPolarity std::function<char(Minisat::Var)> smtrat::VarSchedulerBase::getPolarity [protected], [inherited]

0.15.566.4.11 isAbstractedFormula std::function<bool(const FormulaT&)> smtrat::VarSchedulerBase::isAbstractedFormula [protected], [inherited]

0.15.566.4.12 isBoolValueUndef std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isBoolValueUndef [protected], [inherited]

0.15.566.4.13 isDecisionVar std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isDecisionVar [protected], [inherited]

0.15.566.4.14 isTheoryAbstraction std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isTheoryAbstraction [protected], [inherited]

0.15.566.4.15 isTheoryVar std::function<bool(Minisat::Var)> smtrat::VarSchedulerMcsatBase::isTheoryVar [protected], [inherited]

0.15.566.4.16 minisatVar std::function<Minisat::Var(carl::Variable)> smtrat::VarSchedulerMcsatBase::minisatVar [protected], [inherited]

0.15.566.4.17 reabstractLiteral std::function<const FormulaT&(Minisat::Lit)> smtrat::VarSchedulerBase::reabstractLiteral [protected], [inherited]

0.15.566.4.18 reabstractVariable `std::function<const FormulaT&(Minisat::Var)> smtrat::VarSchedulerBase::reabstractVariable [protected], [inherited]`

0.15.566.4.19 setPolarity `std::function<void(Minisat::Var,bool)> smtrat::VarSchedulerBase::setPolarity [protected], [inherited]`

0.15.567 smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot > Class Template Reference

Decides only Constraints univariate in the current theory variable while the theory ordering is static.
`#include <VarSchedulerMcsat.h>`

Public Member Functions

- template<typename BaseModule >
  VarSchedulerMcsatUnivariateConstraintsOnly (BaseModule &baseModule)
- void rebuild ()
- void insert (Minisat::Var var)
- Minisat::Lit pop ()
- bool empty ()
- void print () const
- void increaseActivity (Minisat::Var var)
- void decreaseActivity (Minisat::Var var)
- void rebuildActivities ()
- template<typename Constraints >
  void rebuildTheoryVars (const Constraints &c)
- void attachClause (Minisat::CRef)
- void detachClause (Minisat::CRef)
- void relocateClauses (std::function< void(Minisat::CRef &)>)

Protected Attributes

- std::function< bool(Minisat::Var)> isTheoryVar
- std::function< carl::Variable(Minisat::Var)> carlVar
- std::function< Minisat::Var(carl::Variable)> minisatVar
- std::function< Model()> currentModel
- std::function< double(Minisat::Var)> getActivity
- std::function< char(Minisat::Var)> getPolarity
- std::function< void(Minisat::Var, bool)> setPolarity
- std::function< bool(Minisat::Var)> isDecisionVar
- std::function< bool(Minisat::Var)> isBoolValueUndef
- std::function< bool(Minisat::Var)> isTheoryAbstraction
- std::function< const FormulaT &(Minisat::Var)> reabstractVariable
- std::function< const FormulaT &(Minisat::Lit)> reabstractLiteral
- std::function< const Minisat::Clause &(Minisat::CRef)> getClause
- std::function< Minisat::lbool(Minisat::Var)> getBoolVarValue
- std::function< Minisat::lbool(Minisat::Lit)> getBoolLitValue
- std::function< unsigned(const FormulaT &)> currentlySatisfiedByBackend
- std::function< Minisat::Var(const FormulaT &)> abstractVariable
- std::function< const Minisat::Lit(const FormulaT &)> abstractLiteral
- std::function< bool(const FormulaT &)> isAbstractedFormula

0.15.567.1 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.

0.15.567.2 Constructor & Destructor Documentation

0.15.567.2.1 VarSchedulerMcsatUnivariateConstraintsOnly()

```
template<int lookahead, mcsat::↩
VariableOrdering vot>
template<typename BaseModule >
smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::VarSchedulerMcsatUnivariateConstraintsO
(
            BaseModule & baseModule )  [inline], [explicit]
```

0.15.567.3 Member Function Documentation

0.15.567.3.1 attachClause()

```
void smtrat::VarSchedulerBase::attachClause (
            Minisat::CRef  )  [inline], [inherited]
```

0.15.567.3.2 decreaseActivity()

```
template<int lookahead, mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::decreaseActivity (
            Minisat::Var var )  [inline]
```

0.15.567.3.3 detachClause()

```
void smtrat::VarSchedulerBase::detachClause (
            Minisat::CRef  )  [inline], [inherited]
```

0.15.567.3.4 empty()

```
template<int lookahead, mcsat::VariableOrdering vot>
bool smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::empty ( )  [inline]
```

0.15.567.3.5 increaseActivity()

```
template<int lookahead, mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::increaseActivity (
            Minisat::Var var )  [inline]
```

0.15.567.3.6 insert()

```
template<int lookahead, mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::insert (
            Minisat::Var var )  [inline]
```

0.15.567.3.7 pop()

```
template<int lookahead, mcsat::VariableOrdering vot>
Minisat::Lit smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::pop ( )
[inline]
```

0.15.567.3.8 print()

```
template<int lookahead, mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::print ( ) const
[inline]
```

0.15.567.3.9 rebuild()

```
template<int lookahead, mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::rebuild ( ) [inline]
```

0.15.567.3.10 rebuildActivities()

```
template<int lookahead, mcsat::VariableOrdering vot>
void smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::rebuildActivities (
) [inline]
```

0.15.567.3.11 rebuildTheoryVars()

```
template<int lookahead, mcsat::VariableOrdering vot>
template<typename Constraints >
void smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::rebuildTheoryVars (
            const Constraints & c ) [inline]
```

0.15.567.3.12 relocateClauses()

```
void smtrat::VarSchedulerBase::relocateClauses (
            std::function< void(Minisat::CRef &)> ) [inline], [inherited]
```

0.15.567.4 Field Documentation

0.15.567.4.1 abstractLiteral

```
std::function<const Minisat::Lit(const FormulaT&)> smtrat::Var↩
SchedulerBase::abstractLiteral [protected], [inherited]
```

0.15.567.4.2 abstractVariable

```
std::function< Minisat::Var(const FormulaT&)> smtrat::VarScheduler↩
Base::abstractVariable [protected], [inherited]
```

0.15.567.4.3 carlVar

```
std::function<carl::Variable(Minisat::Var)> smtrat::VarSchedulerMcsat↩
Base::carlVar [protected], [inherited]
```

0.15.567.4.4 currentlySatisfiedByBackend

```
std::function<unsigned(const FormulaT&)> smtrat::Var↩
SchedulerBase::currentlySatisfiedByBackend [protected], [inherited]
```

0.15.567.4.5 currentModel

```
std::function<Model()> smtrat::VarSchedulerMcsatBase::currentModel
[protected], [inherited]
```

0.15.567.4.6 getActivity

```
std::function<double(Minisat::Var)> smtrat::VarSchedulerBase::get↩
Activity [protected], [inherited]
```

0.15.567.4.7 getBoolLitValue

```
std::function<Minisat::lbool(Minisat::Lit)> smtrat::VarScheduler↩
Base::getBoolLitValue [protected], [inherited]
```

0.15.567.4.8 getBoolVarValue

```
std::function<Minisat::lbool(Minisat::Var)> smtrat::VarScheduler↩
Base::getBoolVarValue [protected], [inherited]
```

0.15.567.4.9 getClause

`std::function<const Minisat::Clause&(Minisat::CRef)> smtrat::VarSchedulerBase::getClause [protected], [inherited]`

0.15.567.4.10 getPolarity

`std::function<char(Minisat::Var)> smtrat::VarSchedulerBase::getPolarity [protected], [inherited]`

0.15.567.4.11 isAbstractedFormula

`std::function<bool(const FormulaT&)> smtrat::VarSchedulerBase::isAbstractedFormula [protected], [inherited]`

0.15.567.4.12 isBoolValueUndef

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isBoolValueUndef [protected], [inherited]`

0.15.567.4.13 isDecisionVar

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isDecisionVar [protected], [inherited]`

0.15.567.4.14 isTheoryAbstraction

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isTheoryAbstraction [protected], [inherited]`

0.15.567.4.15 isTheoryVar

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerMcsatBase::isTheoryVar [protected], [inherited]`

0.15.567.4.16 minisatVar

`std::function<Minisat::Var(carl::Variable)> smtrat::VarSchedulerMcsatBase::minisatVar [protected], [inherited]`

0.15.567.4.17 reabstractLiteral

`std::function<const FormulaT&(Minisat::Lit)> smtrat::VarSchedulerBase::reabstractLiteral [protected], [inherited]`

0.15.567.4.18 reabstractVariable

`std::function<const FormulaT&(Minisat::Var)> smtrat::VarSchedulerBase::reabstractVariable [protected], [inherited]`

0.15.567.4.19 setPolarity

`std::function<void(Minisat::Var,bool)> smtrat::VarSchedulerBase::setPolarity [protected], [inherited]`

0.15.568 smtrat::VarSchedulerMinisat Class Reference

Minisat's activity-based variable scheduling.

`#include <VarScheduler.h>`

Public Member Functions

- template<typename BaseModule >
  VarSchedulerMinisat (BaseModule &baseModule, std::function< bool(Minisat::Var, Minisat::Var)> cmp)
- template<typename BaseModule >
  VarSchedulerMinisat (BaseModule &baseModule)
- void rebuild ()
- void insert (Minisat::Var var)

- Minisat::Var top ()
- Minisat::Lit pop ()
- bool empty ()
- void print () const
- void increaseActivity (Minisat::Var var)
- void decreaseActivity (Minisat::Var var)
- void rebuildActivities ()
- template<typename Constraints >
  void rebuildTheoryVars (const Constraints &)
- void attachClause (Minisat::CRef)
- void detachClause (Minisat::CRef)
- void relocateClauses (std::function< void(Minisat::CRef &)>)

Protected Member Functions

- bool valid (Minisat::Var var)

Protected Attributes

- std::function< double(Minisat::Var)> getActivity
- std::function< char(Minisat::Var)> getPolarity
- std::function< void(Minisat::Var, bool)> setPolarity
- std::function< bool(Minisat::Var)> isDecisionVar
- std::function< bool(Minisat::Var)> isBoolValueUndef
- std::function< bool(Minisat::Var)> isTheoryAbstraction
- std::function< const FormulaT &(Minisat::Var)> reabstractVariable
- std::function< const FormulaT &(Minisat::Lit)> reabstractLiteral
- std::function< const Minisat::Clause &(Minisat::CRef)> getClause
- std::function< Minisat::lbool(Minisat::Var)> getBoolVarValue
- std::function< Minisat::lbool(Minisat::Lit)> getBoolLitValue
- std::function< unsigned(const FormulaT &)> currentlySatisfiedByBackend
- std::function< Minisat::Var(const FormulaT &)> abstractVariable
- std::function< const Minisat::Lit(const FormulaT &)> abstractLiteral
- std::function< bool(const FormulaT &)> isAbstractedFormula

0.15.568.1 Detailed Description

Minisat's activity-based variable scheduling.

0.15.568.2 Constructor & Destructor Documentation

0.15.568.2.1 VarSchedulerMinisat() [1/2]

```
template<typename BaseModule >
smtrat::VarSchedulerMinisat::VarSchedulerMinisat (
            BaseModule & baseModule,
            std::function< bool(Minisat::Var, Minisat::Var)> cmp )  [inline], [explicit]
```

0.15.568.2.2 VarSchedulerMinisat() [2/2]

```
template<typename BaseModule >
smtrat::VarSchedulerMinisat::VarSchedulerMinisat (
            BaseModule & baseModule )  [inline], [explicit]
```

0.15.568.3 Member Function Documentation

0.15.568.3.1 attachClause()

`void smtrat::VarSchedulerBase::attachClause ( Minisat::CRef ) [inline], [inherited]`

0.15.568.3.2 decreaseActivity()

`void smtrat::VarSchedulerMinisat::decreaseActivity ( Minisat::Var var ) [inline]`

0.15.568.3.3 detachClause()

`void smtrat::VarSchedulerBase::detachClause ( Minisat::CRef ) [inline], [inherited]`

0.15.568.3.4 empty()

`bool smtrat::VarSchedulerMinisat::empty ( ) [inline]`

0.15.568.3.5 increaseActivity()

`void smtrat::VarSchedulerMinisat::increaseActivity ( Minisat::Var var ) [inline]`

0.15.568.3.6 insert()

`void smtrat::VarSchedulerMinisat::insert ( Minisat::Var var ) [inline]`

0.15.568.3.7 pop()

`Minisat::Lit smtrat::VarSchedulerMinisat::pop ( ) [inline]`

0.15.568.3.8 print()

`void smtrat::VarSchedulerMinisat::print ( ) const [inline]`

0.15.568.3.9 rebuild()

`void smtrat::VarSchedulerMinisat::rebuild ( ) [inline]`

0.15.568.3.10 rebuildActivities()

`void smtrat::VarSchedulerMinisat::rebuildActivities ( ) [inline]`

0.15.568.3.11 rebuildTheoryVars()

`template<typename Constraints > void smtrat::VarSchedulerBase::rebuildTheoryVars ( const Constraints & ) [inline], [inherited]`

0.15.568.3.12 relocateClauses()

`void smtrat::VarSchedulerBase::relocateClauses ( std::function< void(Minisat::CRef &)> ) [inline], [inherited]`

0.15.568.3.13 top()

`Minisat::Var smtrat::VarSchedulerMinisat::top ( ) [inline]`

0.15.568.3.14 valid()

`bool smtrat::VarSchedulerMinisat::valid ( Minisat::Var var ) [inline], [protected]`

0.15.568.4 Field Documentation

0.15.568.4.1 abstractLiteral

`std::function<const Minisat::Lit(const FormulaT&)> smtrat::VarSchedulerBase::abstractLiteral [protected], [inherited]`

0.15.568.4.2 abstractVariable

`std::function< Minisat::Var(const FormulaT&)> smtrat::VarSchedulerBase::abstractVariable [protected], [inherited]`

0.15.568.4.3 currentlySatisfiedByBackend

`std::function<unsigned(const FormulaT&)> smtrat::VarSchedulerBase::currentlySatisfiedByBackend [protected], [inherited]`

0.15.568.4.4 getActivity

`std::function<double(Minisat::Var)> smtrat::VarSchedulerBase::getActivity [protected], [inherited]`

0.15.568.4.5 getBoolLitValue

`std::function<Minisat::lbool(Minisat::Lit)> smtrat::VarSchedulerBase::getBoolLitValue [protected], [inherited]`

0.15.568.4.6 getBoolVarValue

`std::function<Minisat::lbool(Minisat::Var)> smtrat::VarSchedulerBase::getBoolVarValue [protected], [inherited]`

0.15.568.4.7 getClause

`std::function<const Minisat::Clause&(Minisat::CRef)> smtrat::VarSchedulerBase::getClause [protected], [inherited]`

0.15.568.4.8 getPolarity

`std::function<char(Minisat::Var)> smtrat::VarSchedulerBase::getPolarity [protected], [inherited]`

0.15.568.4.9 isAbstractedFormula

`std::function<bool(const FormulaT&)> smtrat::VarSchedulerBase::isAbstractedFormula [protected], [inherited]`

0.15.568.4.10 isBoolValueUndef

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isBoolValueUndef [protected], [inherited]`

0.15.568.4.11 isDecisionVar

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isDecisionVar [protected], [inherited]`

0.15.568.4.12 isTheoryAbstraction

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isTheoryAbstraction [protected], [inherited]`

0.15.568.4.13 reabstractLiteral

`std::function<const FormulaT&(Minisat::Lit)> smtrat::VarSchedulerBase::reabstractLiteral [protected], [inherited]`

0.15.568.4.14 reabstractVariable

`std::function<const FormulaT&(Minisat::Var)> smtrat::VarSchedulerBase::reabstractVariable [protected], [inherited]`

0.15.568.4.15 setPolarity `std::function<void(Minisat::Var,bool)> smtrat::VarSchedulerBase↩ ::setPolarity [protected], [inherited]`

0.15.569 smtrat::VarSchedulerRandom Class Reference

Random scheduler.
`#include <VarScheduler.h>`

Public Member Functions

- template<typename BaseModule >
  VarSchedulerRandom (BaseModule &baseModule)
- void rebuild ()
- void insert (Minisat::Var var)
- Minisat::Lit pop ()
- bool empty ()
- void print () const
- void increaseActivity (Minisat::Var)
- void decreaseActivity (Minisat::Var)
- void rebuildActivities ()
- template<typename Constraints >
  void rebuildTheoryVars (const Constraints &)
- void attachClause (Minisat::CRef)
- void detachClause (Minisat::CRef)
- void relocateClauses (std::function< void(Minisat::CRef &)>)

Protected Member Functions

- bool valid (Minisat::Var var)

Protected Attributes

- std::function< double(Minisat::Var)> getActivity
- std::function< char(Minisat::Var)> getPolarity
- std::function< void(Minisat::Var, bool)> setPolarity
- std::function< bool(Minisat::Var)> isDecisionVar
- std::function< bool(Minisat::Var)> isBoolValueUndef
- std::function< bool(Minisat::Var)> isTheoryAbstraction
- std::function< const FormulaT &(Minisat::Var)> reabstractVariable
- std::function< const FormulaT &(Minisat::Lit)> reabstractLiteral
- std::function< const Minisat::Clause &(Minisat::CRef)> getClause
- std::function< Minisat::lbool(Minisat::Var)> getBoolVarValue
- std::function< Minisat::lbool(Minisat::Lit)> getBoolLitValue
- std::function< unsigned(const FormulaT &)> currentlySatisfiedByBackend
- std::function< Minisat::Var(const FormulaT &)> abstractVariable
- std::function< const Minisat::Lit(const FormulaT &)> abstractLiteral
- std::function< bool(const FormulaT &)> isAbstractedFormula

0.15.569.1 Detailed Description

Random scheduler.

0.15.569.2 Constructor & Destructor Documentation

0.15.569.2.1 VarSchedulerRandom()

```
template<typename BaseModule >
smtrat::VarSchedulerRandom::VarSchedulerRandom (
            BaseModule & baseModule ) [inline], [explicit]
```

0.15.569.3 Member Function Documentation

0.15.569.3.1 attachClause()

```
void smtrat::VarSchedulerBase::attachClause (
            Minisat::CRef ) [inline], [inherited]
```

0.15.569.3.2 decreaseActivity()

```
void smtrat::VarSchedulerBase::decreaseActivity (
            Minisat::Var ) [inline], [inherited]
```

0.15.569.3.3 detachClause()

```
void smtrat::VarSchedulerBase::detachClause (
            Minisat::CRef ) [inline], [inherited]
```

0.15.569.3.4 empty()

```
bool smtrat::VarSchedulerRandom::empty ( ) [inline]
```

0.15.569.3.5 increaseActivity()

```
void smtrat::VarSchedulerBase::increaseActivity (
            Minisat::Var ) [inline], [inherited]
```

0.15.569.3.6 insert()

```
void smtrat::VarSchedulerRandom::insert (
            Minisat::Var var ) [inline]
```

0.15.569.3.7 pop()

```
Minisat::Lit smtrat::VarSchedulerRandom::pop ( ) [inline]
```

0.15.569.3.8 print()

```
void smtrat::VarSchedulerRandom::print ( ) const [inline]
```

0.15.569.3.9 rebuild()

```
void smtrat::VarSchedulerRandom::rebuild ( ) [inline]
```

0.15.569.3.10 rebuildActivities()

```
void smtrat::VarSchedulerBase::rebuildActivities ( ) [inline],
[inherited]
```

0.15.569.3.11 rebuildTheoryVars()

```
template<typename Constraints >
void smtrat::VarSchedulerBase::rebuildTheoryVars (
            const Constraints & ) [inline], [inherited]
```

0.15.569.3.12 relocateClauses()

```
void smtrat::VarSchedulerBase::relocateClauses (
            std::function< void(Minisat::CRef &)> ) [inline], [inherited]
```

0.15.569.3.13 valid()

```
bool smtrat::VarSchedulerRandom::valid (
            Minisat::Var var ) [inline], [protected]
```

0.15.569.4 Field Documentation

0.15.569.4.1 abstractLiteral

`std::function<const Minisat::Lit(const FormulaT&)> smtrat::VarSchedulerBase::abstractLiteral [protected], [inherited]`

0.15.569.4.2 abstractVariable

`std::function< Minisat::Var(const FormulaT&)> smtrat::VarSchedulerBase::abstractVariable [protected], [inherited]`

0.15.569.4.3 currentlySatisfiedByBackend

`std::function<unsigned(const FormulaT&)> smtrat::VarSchedulerBase::currentlySatisfiedByBackend [protected], [inherited]`

0.15.569.4.4 getActivity

`std::function<double(Minisat::Var)> smtrat::VarSchedulerBase::getActivity [protected], [inherited]`

0.15.569.4.5 getBoolLitValue

`std::function<Minisat::lbool(Minisat::Lit)> smtrat::VarSchedulerBase::getBoolLitValue [protected], [inherited]`

0.15.569.4.6 getBoolVarValue

`std::function<Minisat::lbool(Minisat::Var)> smtrat::VarSchedulerBase::getBoolVarValue [protected], [inherited]`

0.15.569.4.7 getClause

`std::function<const Minisat::Clause&(Minisat::CRef)> smtrat::VarSchedulerBase::getClause [protected], [inherited]`

0.15.569.4.8 getPolarity

`std::function<char(Minisat::Var)> smtrat::VarSchedulerBase::getPolarity [protected], [inherited]`

0.15.569.4.9 isAbstractedFormula

`std::function<bool(const FormulaT&)> smtrat::VarSchedulerBase::isAbstractedFormula [protected], [inherited]`

0.15.569.4.10 isBoolValueUndef

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isBoolValueUndef [protected], [inherited]`

0.15.569.4.11 isDecisionVar

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isDecisionVar [protected], [inherited]`

0.15.569.4.12 isTheoryAbstraction

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isTheoryAbstraction [protected], [inherited]`

0.15.569.4.13 reabstractLiteral

`std::function<const FormulaT&(Minisat::Lit)> smtrat::VarSchedulerBase::reabstractLiteral [protected], [inherited]`

0.15.569.4.14 reabstractVariable `std::function<const FormulaT&(Minisat::Var)> smtrat::VarSchedulerBase::reabstractVariable [protected], [inherited]`

0.15.569.4.15 setPolarity `std::function<void(Minisat::Var,bool)> smtrat::VarSchedulerBase::setPolarity [protected], [inherited]`

0.15.570 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

0.15.570.1 Detailed Description

template<TheoryGuidedDecisionHeuristicLevel theory_conflict_guided_decision_heuristic>
class smtrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic >

Scheduler for SMT, implementing theory guided heuristics.

0.15.570.2 Constructor & Destructor Documentation

0.15.570.2.1 VarSchedulerSMTTheoryGuided()

```
template<TheoryGuidedDecisionHeuristicLevel theory↩
_conflict_guided_decision_heuristic>
template<typename BaseModule >
smtrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic >::VarSchedulerSMTTheoryGuided
(
            BaseModule & baseModule )  [inline], [explicit]
```

0.15.570.3 Member Function Documentation

0.15.570.3.1 attachClause()

```
void smtrat::VarSchedulerBase::attachClause (
            Minisat::CRef  )  [inline], [inherited]
```

0.15.570.3.2 decreaseActivity()

```
template<TheoryGuidedDecisionHeuristicLevel theory_conflict_↩
guided_decision_heuristic>
void smtrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic >::decrease↩
Activity (
            Minisat::Var var )  [inline]
```

0.15.570.3.3 detachClause()

```
void smtrat::VarSchedulerBase::detachClause (
            Minisat::CRef  )  [inline], [inherited]
```

0.15.570.3.4 empty()

```
template<TheoryGuidedDecisionHeuristicLevel theory_conflict_guided_↩
decision_heuristic>
bool smtrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic >::empty (
)  [inline]
```

0.15.570.3.5 increaseActivity()

```
template<TheoryGuidedDecisionHeuristicLevel theory_conflict_↩
guided_decision_heuristic>
void smtrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic >::increase↩
Activity (
            Minisat::Var var )  [inline]
```

0.15.570.3.6 insert()

```
template<TheoryGuidedDecisionHeuristicLevel theory_conflict_guided_↩
decision_heuristic>
void smtrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic >::insert
(
            Minisat::Var var )  [inline]
```

0.15.570.3.7 pop()

```
template<TheoryGuidedDecisionHeuristicLevel theory_conflict_guided_decision↩
_heuristic>
Minisat::Lit smtrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic
>::pop ( )  [inline]
```

0.15.570.3.8 print()

```
template<TheoryGuidedDecisionHeuristicLevel theory_conflict_guided_decision↩
_heuristic>
void smtrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic >::print (
) const [inline]
```

0.15.570.3.9 rebuild()

```
template<TheoryGuidedDecisionHeuristicLevel theory_conflict_guided_↩
decision_heuristic>
void smtrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic >::rebuild
( ) [inline]
```

0.15.570.3.10 rebuildActivities()

```
template<TheoryGuidedDecisionHeuristicLevel theory_conflict_↩
guided_decision_heuristic>
void smtrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic >::rebuild↩
Activities ( ) [inline]
```

0.15.570.3.11 rebuildTheoryVars()

```
template<typename Constraints >
void smtrat::VarSchedulerBase::rebuildTheoryVars (
            const Constraints & ) [inline], [inherited]
```

0.15.570.3.12 relocateClauses()

```
void smtrat::VarSchedulerBase::relocateClauses (
            std::function< void(Minisat::CRef &)> ) [inline], [inherited]
```

0.15.570.4 Field Documentation

0.15.570.4.1 abstractLiteral

```
std::function<const Minisat::Lit(const FormulaT&)> smtrat::Var↩
SchedulerBase::abstractLiteral [protected], [inherited]
```

0.15.570.4.2 abstractVariable

```
std::function< Minisat::Var(const FormulaT&)> smtrat::VarScheduler↩
Base::abstractVariable [protected], [inherited]
```

0.15.570.4.3 currentlySatisfiedByBackend

```
std::function<unsigned(const FormulaT&)> smtrat::Var↩
SchedulerBase::currentlySatisfiedByBackend [protected], [inherited]
```

0.15.570.4.4 getActivity

```
std::function<double(Minisat::Var)> smtrat::VarSchedulerBase::get↩
Activity [protected], [inherited]
```

0.15.570.4.5 getBoolLitValue

```
std::function<Minisat::lbool(Minisat::Lit)> smtrat::VarScheduler↩
Base::getBoolLitValue [protected], [inherited]
```

0.15.570.4.6 getBoolVarValue

```
std::function<Minisat::lbool(Minisat::Var)> smtrat::VarScheduler↩
Base::getBoolVarValue [protected], [inherited]
```

0.15.570.4.7 getClause

`std::function<const Minisat::Clause&(Minisat::CRef)> smtrat::VarSchedulerBase::getClause [protected], [inherited]`

0.15.570.4.8 getPolarity

`std::function<char(Minisat::Var)> smtrat::VarSchedulerBase::getPolarity [protected], [inherited]`

0.15.570.4.9 isAbstractedFormula

`std::function<bool(const FormulaT&)> smtrat::VarSchedulerBase::isAbstractedFormula [protected], [inherited]`

0.15.570.4.10 isBoolValueUndef

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isBoolValueUndef [protected], [inherited]`

0.15.570.4.11 isDecisionVar

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isDecisionVar [protected], [inherited]`

0.15.570.4.12 isTheoryAbstraction

`std::function<bool(Minisat::Var)> smtrat::VarSchedulerBase::isTheoryAbstraction [protected], [inherited]`

0.15.570.4.13 reabstractLiteral

`std::function<const FormulaT&(Minisat::Lit)> smtrat::VarSchedulerBase::reabstractLiteral [protected], [inherited]`

0.15.570.4.14 reabstractVariable

`std::function<const FormulaT&(Minisat::Var)> smtrat::VarSchedulerBase::reabstractVariable [protected], [inherited]`

0.15.570.4.15 setPolarity

`std::function<void(Minisat::Var,bool)> smtrat::VarSchedulerBase::setPolarity [protected], [inherited]`

0.15.571 Minisat::vec< T > Class Template Reference

`#include <Vec.h>`

Public Member Functions

- vec ()
- vec (int size)
- vec (int size, const T &pad)
- vec (vec< T > &&_toMove)
- ~vec ()
- vec< T > & operator= (vec< T > &&_toMove)
- operator T* (void)
- int size (void) const
- void shrink (int nelems)
- void shrink_ (int nelems)
- int capacity (void) const
- void capacity (int min_cap)
- void growTo (int size)
- void growTo (int size, const T &pad)

- void clear (bool dealloc=false)
- void push (void)
- void push (const T &elem)
- void push_ (const T &elem)
- void pop (void)
- const T & last (void) const
- T & last (void)
- const T & operator[ ] (int index) const
- T & operator[ ] (int index)
- void copyTo (vec< T > &copy) const
- void moveTo (vec< T > &dest)

0.15.571.1 Constructor & Destructor Documentation

0.15.571.1.1 vec() [1/4] template<class T >

```
Minisat::vec< T >::vec ( ) [inline]
```

0.15.571.1.2 vec() [2/4] template<class T >

```
Minisat::vec< T >::vec (
            int size ) [inline], [explicit]
```

0.15.571.1.3 vec() [3/4] template<class T >

```
Minisat::vec< T >::vec (
            int size,
            const T & pad ) [inline]
```

0.15.571.1.4 vec() [4/4] template<class T >

```
Minisat::vec< T >::vec (
            vec< T > && _toMove ) [inline]
```

0.15.571.1.5 ~vec() template<class T >

```
Minisat::vec< T >::~vec ( ) [inline]
```

0.15.571.2 Member Function Documentation

0.15.571.2.1 capacity() [1/2] template<class T >

```
void Minisat::vec< T >::capacity (
            int min_cap )
```

0.15.571.2.2 capacity() [2/2] template<class T >

```
int Minisat::vec< T >::capacity (
            void ) const [inline]
```

0.15.571.2.3 clear() template<class T >

```
void Minisat::vec< T >::clear (
            bool dealloc = false )
```

0.15.571.2.4 copyTo() template<class T >

```
void Minisat::vec< T >::copyTo (
            vec< T > & copy ) const  [inline]
```

0.15.571.2.5 growTo() [1/2] template<class T >

```
void Minisat::vec< T >::growTo (
            int size )
```

0.15.571.2.6 growTo() [2/2] template<class T >

```
void Minisat::vec< T >::growTo (
            int size,
            const T & pad )
```

0.15.571.2.7 last() [1/2] template<class T >

```
T& Minisat::vec< T >::last (
            void  )  [inline]
```

0.15.571.2.8 last() [2/2] template<class T >

```
const T& Minisat::vec< T >::last (
            void  ) const  [inline]
```

0.15.571.2.9 moveTo() template<class T >

```
void Minisat::vec< T >::moveTo (
            vec< T > & dest )  [inline]
```

0.15.571.2.10 operator T*() template<class T >

```
Minisat::vec< T >::operator T* (
            void  )  [inline]
```

0.15.571.2.11 operator=() template<class T >

```
vec<T>& Minisat::vec< T >::operator= (
            vec< T > && _toMove )  [inline]
```

0.15.571.2.12 operator[]() [1/2] template<class T >

```
T& Minisat::vec< T >::operator[] (
            int index )  [inline]
```

0.15.571.2.13 operator[]() [2/2] template<class T >

```
const T& Minisat::vec< T >::operator[] (
            int index ) const  [inline]
```

0.15.571.2.14 pop() template<class T >

```
void Minisat::vec< T >::pop (
            void  )  [inline]
```

0.15.571.2.15 push() [1/2] template<class T >

```
void Minisat::vec< T >::push (
            const T & elem )  [inline]
```

0.15.571.2.16 push() [2/2] template<class T >

```
void Minisat::vec< T >::push (
            void  )  [inline]
```

0.15.571.2.17 push_() template<class T >

```
void Minisat::vec< T >::push_ (
            const T & elem )  [inline]
```

0.15.571.2.18 shrink() template<class T >

```
void Minisat::vec< T >::shrink (
            int nelems )  [inline]
```

0.15.571.2.19 shrink_() template<class T >

```
void Minisat::vec< T >::shrink_ (
            int nelems )  [inline]
```

0.15.571.2.20 size() template<class T >

```
int Minisat::vec< T >::size (
            void  ) const  [inline]
```

0.15.572 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

0.15.572.1 Detailed Description

template<typename Res>
struct smtrat::parser::conversion::VectorVariantConverter< Res >

Converts a vector of variants to a vector of some type using the Converter class.

0.15.572.2 Member Typedef Documentation

0.15.572.2.1 result_type

```
template<typename Res >
typedef Res smtrat::parser::conversion::VectorVariantConverter< Res >::result_type
```

0.15.572.3 Member Function Documentation

0.15.572.3.1 operator()() [1/2]

```
template<typename Res >
template<BOOST_VARIANT_ENUM_PARAMS(typename T) >
bool smtrat::parser::conversion::VectorVariantConverter< Res >::operator() (
            const std::vector< boost::variant< BOOST_VARIANT_ENUM_PARAMS(T)>> & v,
            std::vector< Res > & result ) const  [inline]
```

0.15.572.3.2 operator()() [2/2]

```
template<typename Res >
template<BOOST_VARIANT_ENUM_PARAMS(typename T) >
bool smtrat::parser::conversion::VectorVariantConverter< Res >::operator() (
            const std::vector< boost::variant< BOOST_VARIANT_ENUM_PARAMS(T)>> & v,
            std::vector< Res > & result,
            TheoryError & errors ) const  [inline]
```

0.15.573 smtrat::subtropical::Vertex Struct Reference

Represents a term of an original constraint and assigns him a rating variable if a weak separator is searched.

`#include <Subtropical.h>`

Public Member Functions

- Vertex (const TermT &term)
- carl::Variable rating () const

Data Fields

- const Rational coefficient

  Coefficient of the assigned term.
- const carl::Monomial::Arg monomial

  Monomial of the assigned term.
- carl::Variable m_rating

  Rating variable of the term for a weak separator.

0.15.573.1 Detailed Description

Represents a term of an original constraint and assigns him a rating variable if a weak separator is searched.

0.15.573.2 Constructor & Destructor Documentation

0.15.573.2.1 Vertex()

```
smtrat::subtropical::Vertex::Vertex (
            const TermT & term )  [inline]
```

0.15.573.3 Member Function Documentation

0.15.573.3.1 rating()

```
carl::Variable smtrat::subtropical::Vertex::rating ( ) const  [inline]
```

0.15.573.4 Field Documentation

0.15.573.4.1 coefficient `const Rational smtrat::subtropical::Vertex::coefficient`

Coefficient of the assigned term.

0.15.573.4.2 m_rating `carl::Variable smtrat::subtropical::Vertex::m_rating [mutable]`

Rating variable of the term for a weak separator.

0.15.573.4.3 monomial `const carl::Monomial::Arg smtrat::subtropical::Vertex::monomial`

Monomial of the assigned term.

0.15.574 smtrat::VSModule< Settings > Class Template Reference

`#include <VSModule.h>`

Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

Public Member Functions

- std::string moduleName () const
- VSModule (const ModuleInput *, Conditionals &, Manager *const =NULL)
- ~VSModule ()
- bool addCore (ModuleInput::const_iterator)
- Answer checkCore ()

  Checks the received formula for consistency.
- void removeCore (ModuleInput::const_iterator)
- 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 ([[maybe_unused]] const FormulaT &_constraint)

  Informs the module about the given constraint.
- virtual void deinformCore ([[maybe_unused]] const FormulaT &_constraint)

  The inverse of informing about a constraint.
- virtual bool addCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  The module has to take the given sub-formula of the received formula into account.
- virtual void removeCore ([[maybe_unused]] ModuleInput::const_iterator formula)

  Removes everything related to the given sub-formula of the received formula.
- bool anAnswerFound () const

  Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void clearModel () const

  Clears the assignment, if any was found.
- void clearModels () const

  Clears all assignments, if any was found.
- void cleanModel () const

  Substitutes variable occurrences by its model value in the model values of other variables.
- ModuleInput::iterator passedFormulaBegin ()

- ModuleInput::iterator passedFormulaEnd ()
- void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)

  Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const

  Gets the origins of the passed formula at the given position.
- void getOrigins (const FormulaT &_formula, FormulasT &_origins) const
- void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const
- std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)
- std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)
- void informBackends (const FormulaT &_constraint)

  Informs all backends of this module about the given constraint.
- virtual void addConstraintToInform (const FormulaT &_constraint)

  Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.
- std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)

  Copies the given sub-formula of the received formula to the passed formula.
- bool originInReceivedFormula (const FormulaT &_origin) const
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)

  Adds the given formula to the passed formula with no origin.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)

  Adds the given formula to the passed formula.
- std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)

  Adds the given formula to the passed formula.
- void generateTrivialInfeasibleSubset ()

  Stores the trivial infeasible subset being the set of received formulas.
- void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)

  Stores an infeasible subset consisting only of the given received formula.
- std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const
- void getInfeasibleSubsets ()

  Copies the infeasible subsets of the passed formula.
- std::vector< FormulaSetT > getInfeasibleSubsets (const Module &_backend) const

  Get the infeasible subsets the given backend provides.
- const Model & backendsModel () const

  Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void getBackendsModel () const
- void getBackendsAllModels () const

  Stores all models of a backend in the list of all models of this module.
- virtual Answer runBackends (bool _final, bool _full, carl::Variable _objective)

  Runs the backend solvers on the passed formula.
- virtual Answer runBackends ()
- virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)

  Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void clearPassedFormula ()
- std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const

  Merges the two vectors of sets into the first one.
- size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const

- bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const

  Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)

  Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)
- bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())
- void splitUnequalConstraint (const FormulaT &)

  Adds the following lemmas for the given constraint p!=0.
- unsigned checkModel () const
- void addInformationRelevantFormula (const FormulaT &formula)

  Adds a formula to the InformationRelevantFormula.
- const std::set< FormulaT > & getInformationRelevantFormulas ()

  Gets all InformationRelevantFormulas.
- bool isLemmaLevel (LemmaLevel level)

  Checks if current lemma level is greater or equal to given level.

Static Protected Member Functions

- static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)

  Checks whether there is no variable assigned by both models.

Protected Attributes

- std::vector< FormulaSetT > mInfeasibleSubsets

  Stores the infeasible subsets.
- Manager *const mpManager

  A reference to the manager.
- Model mModel

  Stores the assignment of the current satisfiable result, if existent.
- std::vector< Model > mAllModels

  Stores all satisfying assignments.
- bool mModelComputed

  True, if the model has already been computed.
- bool mFinalCheck

  true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool mFullCheck

  false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable mObjectiveVariable

  Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState

  States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic_bool * mBackendsFoundAnswer

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

- Conditionals mFoundAnswer

  Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector< Module * > mUsedBackends

  The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).
- std::vector< Module * > mAllBackends

  The backends of this module which have been used.
- std::vector< Lemma > mLemmas

  Stores the lemmas being valid formulas this module or its backends made.
- ModuleInput::iterator mFirstSubformulaToPass

  Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.
- carl::FastSet< FormulaT > mConstraintsToInform

  Stores the constraints which the backends must be informed about.
- carl::FastSet< FormulaT > mInformedConstraints

  Stores the position of the first constraint of which no backend has been informed about.
- ModuleInput::const_iterator mFirstUncheckedReceivedSubformula

  Stores the position of the first (by this module) unchecked sub-formula of the received formula.
- unsigned mSmallerMusesCheckCounter

  Counter used for the generation of the smt2 files to check for smaller muses.
- std::vector< std::size_t > mVariableCounters

  Maps variables to the number of their occurrences.

0.15.574.1 Member Typedef Documentation

0.15.574.1.1 SettingsType

```
template<class Settings >
typedef Settings smtrat::VSModule< Settings >::SettingsType
```

0.15.574.2 Member Enumeration Documentation

0.15.574.2.1 LemmaType

```
enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

NORMAL	
PERMANENT	

0.15.574.3 Constructor & Destructor Documentation

0.15.574.3.1 VSModule()

```
template<class Settings >
smtrat::VSModule< Settings >::VSModule (
            const ModuleInput * ,
            Conditionals & ,
            Manager * const  = NULL )
```

0.15.574.3.2 ~VSModule() template<class Settings >

smtrat::VSModule< Settings >::~VSModule ( )

0.15.574.4 Member Function Documentation

0.15.574.4.1 add() bool smtrat::Module::add (ModuleInput::const_iterator *_subformula*) [inherited]

The module has to take the given sub-formula of the received formula into account.

Parameters

_subformula	The sub-formula to take additionally into account.

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

0.15.574.4.2 addConstraintToInform() void smtrat::Module::addConstraintToInform (const FormulaT & *_constraint*) [protected], [virtual], [inherited]

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

_constraint	The constraint to add.

Reimplemented in smtrat::SATModule< Settings >.

0.15.574.4.3 addCore() [1/2] virtual bool smtrat::Module::addCore ([[maybe_unused]] 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.

0.15.574.4.4 addCore() [2/2] template<class Settings >

bool smtrat::VSModule< Settings >::addCore (ModuleInput::const_iterator)

0.15.574.4.5 addInformationRelevantFormula() void smtrat::Module::addInformationRelevantFormula (const FormulaT & *formula*) [protected], [inherited]

Adds a formula to the InformationRelevantFormula.

Parameters

formula	Formula to add

0.15.574.4.6 addLemma()

```
void smtrat::Module::addLemma (
    const FormulaT & _lemma,
    const LemmaType & _lt = LemmaType::NORMAL,
    const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline], [inherited]
```

Stores a lemma being a valid formula.

Parameters

_lemma	The eduction/lemma to store.
_lt	The type of the lemma.
_preferredFormula	Hint for the next decision, which formula should be assigned to true.

0.15.574.4.7 addOrigin()

```
void smtrat::Module::addOrigin (
    ModuleInput::iterator _formula,
    const FormulaT & _origin ) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

Parameters

_formula	The passed formula to set the origins for.
_origin	A set of formulas in the received formula of this module.

0.15.574.4.8 addReceivedSubformulaToPassedFormula()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::addReceivedSubformulaToPassedFormula (
    ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.
Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

Parameters

_subformula	The sub-formula of the received formula to copy.

0.15.574.4.9 addSubformulaToPassedFormula() [1/3]

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::addSubformulaToPassedFormula (
    const FormulaT & _formula ) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

Parameters

_formula	The formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.574.4.10 addSubformulaToPassedFormula() [2/3]

```
std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
            const FormulaT & _formula,
            const FormulaT & _origin )  [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

Parameters

_formula	The formula to add to the passed formula.
_origin	The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula.

Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

0.15.574.4.11 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.

0.15.574.4.12 allModels()

```
const std::vector<Model>& smtrat::Module::allModels ( ) const  [inline],
[inherited]
```

Returns

All satisfying assignments, if existent.

0.15.574.4.13 anAnswerFound()

`bool smtrat::Module::anAnswerFound ( ) const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

Returns

True, if one of them has determined a result.

0.15.574.4.14 answerFound()

`const smtrat::Conditionals& smtrat::Module::answerFound ( ) const [inline], [inherited]`

Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.574.4.15 backendsModel()

`const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

0.15.574.4.16 branchAt() [1/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.574.4.17 branchAt() [2/4]

```
bool smtrat::Module::branchAt (
    carl::Variable::Arg _var,
    const Rational & _value,
    std::vector< FormulaT > && _premise,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]
```

0.15.574.4.18 branchAt() [3/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
    bool _leftCaseWeak = true,
    bool _preferLeftCase = true,
    bool _useReceivedFormulaAsPremise = false,
    const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]
```

0.15.574.4.19 branchAt() [4/4]

```
bool smtrat::Module::branchAt (
    const Poly & _polynomial,
    bool _integral,
    const Rational & _value,
```

```
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false )  [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
Note that the given value is rounded down and up, if the given variable is integer-valued.

Parameters

_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

0.15.574.4.20 check()

```
Answer smtrat::Module::check (
bool _final = false,
bool _full = true,
carl::Variable _objective = carl::Variable::NO_VARIABLE )  [virtual], [inherited]
```

Checks the received formula for consistency.
Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

Parameters

_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.

0.15.574.4.21 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.

0.15.574.4.22 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.

0.15.574.4.23 checkModel()

```
unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]
```

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

0.15.574.4.24 cleanModel()

```
void smtrat::Module::cleanModel ( ) const [inline], [protected],
[inherited]
```

Substitutes variable occurrences by its model value in the model values of other variables.

0.15.574.4.25 clearLemmas()

```
void smtrat::Module::clearLemmas ( ) [inline], [inherited]
```

Deletes all yet found lemmas.

0.15.574.4.26 clearModel()

```
void smtrat::Module::clearModel ( ) const [inline], [protected],
[inherited]
```

Clears the assignment, if any was found.

0.15.574.4.27 clearModels()

```
void smtrat::Module::clearModels ( ) const [inline], [protected],
[inherited]
```

Clears all assignments, if any was found.

0.15.574.4.28 clearPassedFormula()

```
void smtrat::Module::clearPassedFormula ( ) [protected],
[inherited]
```

0.15.574.4.29 collectOrigins() [1/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulaSetT & _origins ) const [inherited]
```

0.15.574.4.30 collectOrigins() [2/2]

```
void smtrat::Module::collectOrigins (
    const FormulaT & _formula,
    FormulasT & _origins ) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.
If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

_formula	The formula from which to collect the formulas being sub-formulas of the received formula (origins).
_origins	The set in which to store the origins.

0.15.574.4.31 collectTheoryPropagations()

```
void smtrat::Module::collectTheoryPropagations ( ) [inherited]
```

0.15.574.4.32 constraintsToInform()

```
const carl::FastSet<FormulaT>& smtrat::Module::constraints↩
ToInform ( ) const [inline], [inherited]
```

Returns

The constraints which the backends must be informed about.

0.15.574.4.33 currentlySatisfied()

```
virtual unsigned smtrat::Module::currentlySatisfied (
    const FormulaT & ) const [inline], [virtual], [inherited]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, and smtrat::LRAModule< LRASettings

0.15.574.4.34 currentlySatisfiedByBackend()

```
unsigned smtrat::Module::currentlySatisfiedByBackend (
    const FormulaT & _formula ) const [inherited]
```

Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

0.15.574.4.35 deinform()

```
void smtrat::Module::deinform (
    const FormulaT & _constraint ) [inherited]
```

The inverse of informing about a constraint.
All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

Parameters

_constraint	The constraint to remove from internal data structures.

0.15.574.4.36 deinformCore()

`virtual void smtrat::Module::deinformCore ( [[maybe_unused] ] 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.

0.15.574.4.37 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

0.15.574.4.38 eraseSubformulaFromPassedFormula()

`ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula ( ModuleInput::iterator _subformula, bool _ignoreOrigins = false ) [protected], [virtual], [inherited]`

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

Parameters

_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 >.

0.15.574.4.39 excludeNotReceivedVariablesFromModel()

`void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

0.15.574.4.40 findBestOrigin()

`std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin ( const std::vector< FormulaT > & _origins ) const [protected], [inherited]`

Parameters

_origins	

Returns

0.15.574.4.41 firstSubformulaToPass()

`ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.574.4.42 firstUncheckedReceivedSubformula()

`ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

0.15.574.4.43 freeSplittingVariable()

`static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

0.15.574.4.44 generateTrivialInfeasibleSubset()

`void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

0.15.574.4.45 getBackendsAllModels()

`void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

0.15.574.4.46 getBackendsModel()

`void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

0.15.574.4.47 getInfeasibleSubsets() [1/2]

`void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

0.15.574.4.48 getInfeasibleSubsets() [2/2]

```
std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets (
            const Module & _backend ) const  [protected], [inherited]
```

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

Parameters

_backend	The backend from which to obtain the infeasible subsets.

Returns

The infeasible subsets the given backend provides.

0.15.574.4.49 getInformationRelevantFormulas()

`const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

Returns

Set of all formulas

0.15.574.4.50 getModelEqualities()

`std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

Returns

Equivalence classes.

0.15.574.4.51 getOrigins() [1/3]

```
void smtrat::Module::getOrigins (
const FormulaT & _formula,
FormulaSetT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.574.4.52 getOrigins() [2/3]

```
void smtrat::Module::getOrigins (
const FormulaT & _formula,
FormulasT & _origins ) const [inline], [protected], [inherited]
```

Parameters

_formula	
_origins	

0.15.574.4.53 getOrigins() [3/3]

```
const FormulaT& smtrat::Module::getOrigins (
ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]
```

Gets the origins of the passed formula at the given position.

Parameters

_formula	The position of a formula in the passed formulas.

Returns

The origins of the passed formula at the given position.

0.15.574.4.54 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.

0.15.574.4.55 hasLemmas() `bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

0.15.574.4.56 hasValidInfeasibleSubset() `bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]`

Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

0.15.574.4.57 id() `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

Returns

A unique ID to identify this module instance.

0.15.574.4.58 infeasibleSubsets() `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets ( ) const [inline], [inherited]`

Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas.

0.15.574.4.59 inform() `bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.
It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

Parameters

_constraint	The constraint to inform about.

Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

0.15.574.4.60 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.

0.15.574.4.61 informCore() `virtual bool smtrat::Module::informCore ( [[maybe_unused] ] 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.

0.15.574.4.62 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.

0.15.574.4.63 init() `void smtrat::Module::init ( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.
This method must not be called twice and only before the first runBackends call.
Reimplemented in smtrat::PBPPModule< Settings >, smtrat::PBGaussModule< Settings >, smtrat::NRAILModule< Settings >, smtrat::NewCoveringModule< Settings >, smtrat::NewCADModule< Settings >, smtrat::LRAModule< Settings >, smtrat::LRAModule< LRASettingsICP >, smtrat::LRAModule< LRASettings1 >, smtrat::IntBlastModule< Settings >, smtrat::FPPModule< Settings >, smtrat::CurryModule< Settings >, and smtrat::BVModule< Settings >.

0.15.574.4.64 is_minimizing() `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

0.15.574.4.65 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.

0.15.574.4.66 isPreprocessor() `bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with getReceivedFormulaSimplified().

0.15.574.4.67 lemmas() `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

0.15.574.4.68 merge() `std::vector< FormulaT > smtrat::Module::merge ( const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.
({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

Parameters

_vecSetA	A vector of sets of constraints.
_vecSetB	A vector of sets of constraints.

Returns

The vector being the two given vectors merged.

0.15.574.4.69 model() `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

0.15.574.4.70 modelsDisjoint() `bool smtrat::Module::modelsDisjoint ( const Model & _modelA, const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

Parameters

_modelA	The first model to check for.
_modelB	The second model to check for.

Returns

true, if there is no variable assigned by both models; false, otherwise.

0.15.574.4.71 moduleName()

`template<class Settings >`
`std::string smtrat::VSModule< Settings >::moduleName ( ) const [inline], [virtual]`

Returns

The name of the given module type as name.

Reimplemented from smtrat::Module.

0.15.574.4.72 objective()

`carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

0.15.574.4.73 originInReceivedFormula()

`bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

0.15.574.4.74 passedFormulaBegin()

`ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula. TODO: disable this method

0.15.574.4.75 passedFormulaEnd()

`ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

Returns

An iterator to the end of the passed formula.

0.15.574.4.76 pPassedFormula()

`const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to the backends of this module.

0.15.574.4.77 pReceivedFormula()

`const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

Returns

A pointer to the formula passed to this module.

0.15.574.4.78 print()

`void smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

Parameters

_initiation	The line initiation.

0.15.574.4.79 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.

0.15.574.4.80 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.

0.15.574.4.81 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.

0.15.574.4.82 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.

0.15.574.4.83 printInfeasibleSubsets()

```
void smtrat::Module::printInfeasibleSubsets (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the infeasible subsets.

Parameters

_initiation	The line initiation.

0.15.574.4.84 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.

0.15.574.4.85 printPassedFormula()

```
void smtrat::Module::printPassedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of passed formula.

Parameters

_initiation	The line initiation.

0.15.574.4.86 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.

0.15.574.4.87 printReceivedFormula()

```
void smtrat::Module::printReceivedFormula (
    const std::string & _initiation = "***" ) const [inherited]
```

Prints the vector of the received formula.

Parameters

_initiation	The line initiation.

0.15.574.4.88 probablyLooping()

```
bool smtrat::Module::probablyLooping (
    const typename Poly::PolyType & _branchingPolynomial,
```

```
const Rational & _branchingValue ) const [protected], [inherited]
```

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

_branchingPolynomial	The polynomial to branch at (in most branching strategies this is just a variable).
_branchingValue	The value to branch at.

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

0.15.574.4.89 receivedFormulaChecked()

`void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

0.15.574.4.90 receivedFormulasAsInfeasibleSubset()

```
void smtrat::Module::receivedFormulasAs↩InfeasibleSubset (
ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

0.15.574.4.91 receivedVariable()

```
bool smtrat::Module::receivedVariable (
carl::Variable::Arg _var ) const [inline], [inherited]
```

0.15.574.4.92 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.

0.15.574.4.93 removeCore() [1/2]

```
virtual void smtrat::Module::removeCore (
[[maybe_unused] ] 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.

0.15.574.4.94 removeCore() [2/2]

```
template<class Settings >
void smtrat::VSModule< Settings >::removeCore (
            ModuleInput::const_iterator  )
```

0.15.574.4.95 removeOrigin()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origin (
            ModuleInput::iterator _formula,
            const FormulaT & _origin )  [inline], [protected], [inherited]
```

0.15.574.4.96 removeOrigins()

```
std::pair<ModuleInput::iterator,bool> smtrat::Module::remove↩
Origins (
            ModuleInput::iterator _formula,
            const std::shared_ptr< std::vector< FormulaT >> & _origins )  [inline], [protected],
[inherited]
```

0.15.574.4.97 rPassedFormula()

```
const ModuleInput& smtrat::Module::rPassedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to the backends of this module.

0.15.574.4.98 rReceivedFormula()

```
const ModuleInput& smtrat::Module::rReceivedFormula ( ) const
[inline], [inherited]
```

Returns

A reference to the formula passed to this module.

0.15.574.4.99 runBackends() [1/2]

```
virtual Answer smtrat::Module::runBackends ( )  [inline],
[protected], [virtual], [inherited]
```

Reimplemented in smtrat::PModule.

0.15.574.4.100 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.

0.15.574.4.101 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.

0.15.574.4.102 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.

0.15.574.4.103 solverState() `Answer smtrat::Module::solverState ( ) const [inline], [inherited]`

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

0.15.574.4.104 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 !=.

0.15.574.4.105 threadPriority() `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

0.15.574.4.106 updateAllModels() `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented in smtrat::SATModule< Settings >.

0.15.574.4.107 updateLemmas() `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

0.15.574.4.108 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.

0.15.574.4.109 usedBackends() `const std::vector<Module*>& smtrat::Module::usedBackends ( ) const [inline], [inherited]`

Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.574.5 Field Documentation

0.15.574.5.1 mAllBackends `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

0.15.574.5.2 mAllModels `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

0.15.574.5.3 mBackendsFoundAnswer `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

0.15.574.5.4 mConstraintsToInform `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

0.15.574.5.5 mFinalCheck `bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

0.15.574.5.6 mFirstPosInLastBranches `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

0.15.574.5.7 mFirstSubformulaToPass `ModuleInput::iterator smtrat::Module::mFirstSubformulaTo↩Pass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

0.15.574.5.8 mFirstUncheckedReceivedSubformula

`ModuleInput::const_iterator smtrat::Module::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

0.15.574.5.9 mFoundAnswer

`Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

0.15.574.5.10 mFullCheck

`bool smtrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

0.15.574.5.11 mInfeasibleSubsets

`std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

0.15.574.5.12 mInformedConstraints

`carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`

Stores the position of the first constraint of which no backend has been informed about.

0.15.574.5.13 mLastBranches

`std::vector< Branching > smtrat::Module::mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

0.15.574.5.14 mLemmas

`std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`

Stores the lemmas being valid formulas this module or its backends made.

0.15.574.5.15 mModel

`Model smtrat::Module::mModel [mutable], [protected], [inherited]`

Stores the assignment of the current satisfiable result, if existent.

0.15.574.5.16 mModelComputed

`bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`

True, if the model has already been computed.

0.15.574.5.17 mNumOfBranchVarsToStore

`std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`

The number of different variables to consider for a probable infinite loop of branchings.

0.15.574.5.18 mObjectiveVariable

`carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`

Objective variable to be minimized. If set to NO_VARIABLE, a normal sat check should be performed.

0.15.574.5.19 mOldSplittingVariables

`std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]`

Reusable splitting variables.

0.15.574.5.20 mpManager `Manager* const smtrat::Module::mpManager [protected], [inherited]`

A reference to the manager.

0.15.574.5.21 mSmallerMusesCheckCounter `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`

Counter used for the generation of the smt2 files to check for smaller muses.

0.15.574.5.22 mSolverState `std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

0.15.574.5.23 mTheoryPropagations `std::vector<TheoryPropagation> smtrat::Module::mTheory↩ Propagations [protected], [inherited]`

0.15.574.5.24 mUsedBackends `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

0.15.574.5.25 mVariableCounters `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`

Maps variables to the number of their occurrences.

0.15.575 Minisat::Watcher Struct Reference

[Minisat related code]

`#include <SolverTypes.h>`

Public Member Functions

- Watcher (Minisat::CRef cr, Minisat::Lit p)

  [Minisat related code]
- bool operator== (const Watcher &w) const

  [Minisat related code]
- bool operator!= (const Watcher &w) const

  [Minisat related code]

Data Fields

- Minisat::CRef cref

  [Minisat related code]
- Minisat::Lit blocker

  [Minisat related code]

Friends

- std::ostream & operator<< (std::ostream &os, const Watcher &w)

0.15.575.1 Detailed Description

[Minisat related code]

0.15.575.2 Constructor & Destructor Documentation

0.15.575.2.1 Watcher()

```
Minisat::Watcher::Watcher (
    Minisat::CRef cr,
    Minisat::Lit p ) [inline]
```

[Minisat related code]

0.15.575.3 Member Function Documentation

0.15.575.3.1 operator"!=()

```
bool Minisat::Watcher::operator!= (
    const Watcher & w ) const [inline]
```

[Minisat related code]

0.15.575.3.2 operator==()

```
bool Minisat::Watcher::operator== (
    const Watcher & w ) const [inline]
```

[Minisat related code]

0.15.575.4 Friends And Related Function Documentation

0.15.575.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const Watcher & w ) [friend]
```

0.15.575.5 Field Documentation

0.15.575.5.1 blocker

```
Minisat::Lit Minisat::Watcher::blocker
```

[Minisat related code]

0.15.575.5.2 cref

```
Minisat::CRef Minisat::Watcher::cref
```

[Minisat related code]

0.15.576 smtrat::ICPModule< Settings >::weights Struct Reference

```
#include <ICPModule.h>
```

Data Fields

- std::list< linearVariable * > origins
- double weight

0.15.576.1 Field Documentation

0.15.576.1.1 origins

```
template<class Settings >
std::list<linearVariable*> smtrat::ICPModule< Settings >::weights::origins
```

0.15.576.1.2 weight `template<class Settings >`
`double smtrat::ICPModule< Settings >::weights::weight`

0.15.577 benchmax::XMLWriter Class Reference

Writes results to a xml file.
`#include <XMLWriter.h>`

Public Member Functions

- XMLWriter (const std::string &filename)

  Open file for writing.
- template<typename Results >
  void write (const Jobs &jobs, const Results &results)

  Write results to file.

0.15.577.1 Detailed Description

Writes results to a xml file.

0.15.577.2 Constructor & Destructor Documentation

0.15.577.2.1 XMLWriter() `benchmax::XMLWriter::XMLWriter ( const std::string & filename ) [inline]`
Open file for writing.

0.15.577.3 Member Function Documentation

0.15.577.3.1 write() `template<typename Results >`
`void benchmax::XMLWriter::write ( const Jobs & jobs, const Results & results ) [inline]`
Write results to file.

0.15.578 smtrat::covering_ng::formula::node_ds::XOR Struct Reference

`#include <FormulaEvaluation.h>`

Data Fields

- std::vector< Node > subformulas

0.15.578.1 Field Documentation

0.15.578.1.1 subformulas `std::vector<Node> smtrat::covering_ng::formula::node_ds::XOR::subformulas`

0.15.579 benchmax::Z3 Class Reference

Tool wrapper for Z3 for SMT-LIB.
`#include <Z3.h>`

Public Member Functions

- Z3 (const fs::path &binary, const std::string &arguments)

  New Z3 tool.
- virtual bool canHandle (const fs::path &path) const override

  Only handle .smt2 files.
- virtual void additionalResults (const fs::path &, BenchmarkResult &result) const override

  Parse answer string from stdout.
- std::string name () const

  Common name of this tool.
- fs::path binary () const

  Full path to the binary.
- const std::map< std::string, std::string > & attributes () const

  A set of attributes, for example compilation options.
- std::vector< std::string > resolveDependencies () const

  Get dependencies of binary required to run it (via ldd)
- std::size_t attributeHash () const

  Hash of the attributes.
- virtual std::string getCommandline (const std::string &file) const

  Compose commandline for this tool and the given input file.
- virtual std::string getCommandline (const std::string &file, const std::string &localBinary) const

  Compose commandline for this tool with another binary name and the given input file.
- virtual std::optional< std::string > parseCommandline (const std::string &cmdline) const

  Compose commandline for this tool and the given input file.

Protected Attributes

- std::string mName

  (Non-unique) identifier for the tool.
- fs::path mBinary

  Path to the binary.
- std::string mArguments

  Command line arguments that should be passed to the binary.
- std::map< std::string, std::string > mAttributes

  Attributes of the tool obtained by introspection of the binary.

0.15.579.1 Detailed Description

Tool wrapper for Z3 for SMT-LIB.

0.15.579.2 Constructor & Destructor Documentation

0.15.579.2.1 Z3()

```
benchmax::Z3::Z3 (
    const fs::path & binary,
    const std::string & arguments ) [inline]
```

New Z3 tool.

0.15.579.3 Member Function Documentation

0.15.579.3.1 additionalResults()

```
virtual void benchmax::Z3::additionalResults (
    const fs::path & ,
    BenchmarkResult & result ) const [inline], [override], [virtual]
```

Parse answer string from stdout.
Reimplemented from benchmax::Tool.

0.15.579.3.2 attributeHash()

```
std::size_t benchmax::Tool::attributeHash ( ) const [inline], [inherited]
```

Hash of the attributes.

0.15.579.3.3 attributes()

```
const std::map<std::string,std::string>& benchmax::Tool::attributes ( ) const [inline], [inherited]
```

A set of attributes, for example compilation options.

0.15.579.3.4 binary()

```
fs::path benchmax::Tool::binary ( ) const [inline], [inherited]
```

Full path to the binary.

0.15.579.3.5 canHandle()

```
virtual bool benchmax::Z3::canHandle (
    const fs::path & path ) const [inline], [override], [virtual]
```

Only handle .smt2 files.
Reimplemented from benchmax::Tool.

0.15.579.3.6 getCommandline() [1/2]

```
virtual std::string benchmax::Tool::getCommandline (
    const std::string & file ) const [inline], [virtual], [inherited]
```

Compose commandline for this tool and the given input file.

0.15.579.3.7 getCommandline() [2/2]

```
virtual std::string benchmax::Tool::getCommandline (
    const std::string & file,
    const std::string & localBinary ) const [inline], [virtual], [inherited]
```

Compose commandline for this tool with another binary name and the given input file.

0.15.579.3.8 name()

```
std::string benchmax::Tool::name ( ) const [inline], [inherited]
```

Common name of this tool.

0.15.579.3.9 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.

0.15.579.3.10 resolveDependencies()

```
std::vector<std::string> benchmax::Tool::resolveDependencies ( ) const [inline], [inherited]
```

Get dependencies of binary required to run it (via ldd)

0.15.579.4 Field Documentation

0.15.579.4.1 mArguments `std::string benchmax::Tool::mArguments [protected], [inherited]`

Command line arguments that should be passed to the binary.

0.15.579.4.2 mAttributes `std::map<std::string,std::string> benchmax::Tool::mAttributes [protected], [inherited]`

Attributes of the tool obtained by introspection of the binary.

0.15.579.4.3 mBinary `fs::path benchmax::Tool::mBinary [protected], [inherited]`

Path to the binary.

0.15.579.4.4 mName `std::string benchmax::Tool::mName [protected], [inherited]`

(Non-unique) identifier for the tool.