

stv_v2

Generated by Doxygen 1.9.1

1 STV2 - StraTegic Verifier 2	1
1.1 Usage	1
1.2 Tests	1
1.3 Performance estimation	2
1.4 Specification	2
1.5 Examples and templates	3
1.6 Misc	3
2 Hierarchical Index	5
2.1 Class Hierarchy	5
3 Class Index	7
3.1 Class List	7
4 File Index	9
4.1 File List	9
5 Class Documentation	11
5.1 Agent Class Reference	11
5.1.1 Detailed Description	11
5.1.2 Constructor & Destructor Documentation	12
5.1.2.1 Agent()	12
5.1.3 Member Function Documentation	12
5.1.3.1 includesState()	12
5.2 AgentTemplate Class Reference	12
5.2.1 Detailed Description	13
5.2.2 Member Function Documentation	13
5.2.2.1 addInitial()	13
5.2.2.2 addLocal()	14
5.2.2.3 addPersistent()	14
5.2.2.4 addTransition()	15
5.2.2.5 generateAgent()	15
5.2.2.6 setIdent()	16
5.2.2.7 setInitState()	17
5.3 Assignment Class Reference	17
5.3.1 Detailed Description	18
5.3.2 Constructor & Destructor Documentation	18
5.3.2.1 Assignment()	18
5.3.3 Member Function Documentation	18
5.3.3.1 assign()	18
5.4 Cfg Struct Reference	18
5.5 Condition Struct Reference	19
5.5.1 Detailed Description	19
5.6 DotGraph Class Reference	20

5.6.1 Member Function Documentation	20
5.6.1.1 addEdge()	20
5.6.1.2 addNode()	21
5.6.1.3 saveToFile()	21
5.6.2 Member Data Documentation	21
5.6.2.1 styleString	21
5.7 EpistemicClass Struct Reference	22
5.7.1 Detailed Description	22
5.8 ExprAdd Class Reference	22
5.8.1 Detailed Description	23
5.8.2 Constructor & Destructor Documentation	23
5.8.2.1 ExprAdd()	23
5.8.3 Member Function Documentation	23
5.8.3.1 eval()	23
5.9 ExprAnd Class Reference	24
5.9.1 Detailed Description	24
5.9.2 Constructor & Destructor Documentation	24
5.9.2.1 ExprAnd()	24
5.9.3 Member Function Documentation	25
5.9.3.1 eval()	25
5.10 ExprConst Class Reference	25
5.10.1 Detailed Description	25
5.10.2 Constructor & Destructor Documentation	26
5.10.2.1 ExprConst()	26
5.10.3 Member Function Documentation	26
5.10.3.1 eval()	26
5.11 ExprDiv Class Reference	26
5.11.1 Detailed Description	27
5.11.2 Constructor & Destructor Documentation	27
5.11.2.1 ExprDiv()	27
5.11.3 Member Function Documentation	27
5.11.3.1 eval()	27
5.12 ExprEq Class Reference	28
5.12.1 Detailed Description	28
5.12.2 Constructor & Destructor Documentation	28
5.12.2.1 ExprEq()	28
5.12.3 Member Function Documentation	29
5.12.3.1 eval()	29
5.13 ExprGe Class Reference	29
5.13.1 Detailed Description	29
5.13.2 Constructor & Destructor Documentation	30
5.13.2.1 ExprGe()	30

5.13.3 Member Function Documentation	30
5.13.3.1 eval()	30
5.14 ExprGt Class Reference	31
5.14.1 Detailed Description	31
5.14.2 Constructor & Destructor Documentation	31
5.14.2.1 ExprGt()	31
5.14.3 Member Function Documentation	31
5.14.3.1 eval()	31
5.15 ExprHart Class Reference	33
5.15.1 Detailed Description	33
5.15.2 Constructor & Destructor Documentation	33
5.15.2.1 ExprHart()	33
5.15.3 Member Function Documentation	34
5.15.3.1 eval()	34
5.16 ExprIdent Class Reference	34
5.16.1 Detailed Description	35
5.16.2 Constructor & Destructor Documentation	35
5.16.2.1 ExprIdent()	35
5.16.3 Member Function Documentation	35
5.16.3.1 eval() [1/2]	35
5.16.3.2 eval() [2/2]	35
5.17 ExprKnow Class Reference	37
5.17.1 Detailed Description	37
5.17.2 Constructor & Destructor Documentation	37
5.17.2.1 ExprKnow()	37
5.17.3 Member Function Documentation	38
5.17.3.1 eval()	38
5.18 ExprLe Class Reference	38
5.18.1 Detailed Description	39
5.18.2 Constructor & Destructor Documentation	39
5.18.2.1 ExprLe()	39
5.18.3 Member Function Documentation	39
5.18.3.1 eval()	39
5.19 ExprLt Class Reference	40
5.19.1 Detailed Description	40
5.19.2 Constructor & Destructor Documentation	40
5.19.2.1 ExprLt()	40
5.19.3 Member Function Documentation	40
5.19.3.1 eval()	40
5.20 ExprMul Class Reference	42
5.20.1 Detailed Description	42
5.20.2 Constructor & Destructor Documentation	42

5.20.2.1 ExprMul()	42
5.20.3 Member Function Documentation	43
5.20.3.1 eval()	43
5.21 ExprNe Class Reference	43
5.21.1 Detailed Description	44
5.21.2 Constructor & Destructor Documentation	44
5.21.2.1 ExprNe()	44
5.21.3 Member Function Documentation	44
5.21.3.1 eval()	44
5.22 ExprNode Class Reference	45
5.22.1 Detailed Description	45
5.22.2 Member Function Documentation	45
5.22.2.1 eval()	45
5.23 ExprNot Class Reference	46
5.23.1 Detailed Description	46
5.23.2 Constructor & Destructor Documentation	46
5.23.2.1 ExprNot()	46
5.23.3 Member Function Documentation	46
5.23.3.1 eval()	46
5.24 ExprOr Class Reference	47
5.24.1 Detailed Description	47
5.24.2 Constructor & Destructor Documentation	47
5.24.2.1 ExprOr()	47
5.24.3 Member Function Documentation	48
5.24.3.1 eval()	48
5.25 ExprRem Class Reference	48
5.25.1 Detailed Description	49
5.25.2 Constructor & Destructor Documentation	49
5.25.2.1 ExprRem()	49
5.25.3 Member Function Documentation	49
5.25.3.1 eval()	49
5.26 ExprSub Class Reference	50
5.26.1 Detailed Description	50
5.26.2 Constructor & Destructor Documentation	50
5.26.2.1 ExprSub()	50
5.26.3 Member Function Documentation	50
5.26.3.1 eval()	50
5.27 Formula Struct Reference	52
5.28 FormulaTemplate Struct Reference	52
5.28.1 Detailed Description	52
5.29 GlobalModel Struct Reference	53
5.29.1 Detailed Description	53

5.30 GlobalModelGenerator Class Reference	53
5.30.1 Detailed Description	55
5.30.2 Member Function Documentation	55
5.30.2.1 computeEpistemicClassHash()	55
5.30.2.2 computeGlobalStateHash()	56
5.30.2.3 expandAndReduceAllStates()	56
5.30.2.4 expandState()	56
5.30.2.5 expandStateAndReturn()	57
5.30.2.6 findGlobalStateInEpistemicClass()	57
5.30.2.7 findOrCreateEpistemicClass()	57
5.30.2.8 findOrCreateEpistemicClassForKnowledge()	58
5.30.2.9 generateGlobalTransitions()	58
5.30.2.10 generateInitState()	58
5.30.2.11 generateStateFromLocalStates()	59
5.30.2.12 getAgentInstanceByName()	59
5.30.2.13 getCurrentGlobalModel()	59
5.30.2.14 getFormula()	60
5.30.2.15 getFormulaCorectness()	60
5.30.2.16 getFormulaSize()	60
5.30.2.17 initModel()	60
5.31 GlobalState Struct Reference	61
5.31.1 Detailed Description	62
5.31.2 Member Function Documentation	62
5.31.2.1 getGlobalStateEnvironment()	62
5.31.2.2 toString()	62
5.32 GlobalModelGenerator::GlobalStateTupleHash Struct Reference	62
5.33 GlobalTransition Struct Reference	63
5.33.1 Detailed Description	63
5.34 HistoryDbg Class Reference	63
5.34.1 Detailed Description	64
5.34.2 Member Function Documentation	64
5.34.2.1 addEntry()	64
5.34.2.2 cloneEntry()	64
5.34.2.3 markEntry()	65
5.34.2.4 print()	65
5.35 HistoryEntry Struct Reference	65
5.35.1 Detailed Description	66
5.35.2 Member Function Documentation	66
5.35.2.1 toString()	66
5.36 LocalModels Struct Reference	67
5.36.1 Detailed Description	67
5.37 LocalState Class Reference	67

5.37.1 Detailed Description	68
5.37.2 Member Function Documentation	68
5.37.2.1 compare()	68
5.37.2.2 toString()	68
5.38 LocalStateTemplate Class Reference	69
5.38.1 Detailed Description	69
5.39 LocalTransition Struct Reference	69
5.39.1 Detailed Description	70
5.40 ModelParser Class Reference	70
5.40.1 Detailed Description	70
5.40.2 Member Function Documentation	70
5.40.2.1 parse()	70
5.41 TestVerif Class Reference	71
5.42 TransitionTemplate Class Reference	71
5.42.1 Detailed Description	72
5.42.2 Constructor & Destructor Documentation	72
5.42.2.1 TransitionTemplate()	72
5.43 Var Struct Reference	73
5.43.1 Detailed Description	73
5.44 Verification Class Reference	73
5.44.1 Detailed Description	75
5.44.2 Constructor & Destructor Documentation	75
5.44.2.1 Verification()	75
5.44.3 Member Function Documentation	75
5.44.3.1 addHistoryContext()	75
5.44.3.2 addHistoryDecision()	76
5.44.3.3 addHistoryMarkDecisionAsInvalid()	76
5.44.3.4 addHistoryStateStatus()	76
5.44.3.5 areGlobalStatesInTheSameEpistemicClass()	77
5.44.3.6 checkUncontrolledSet()	77
5.44.3.7 equivalentGlobalTransitions()	78
5.44.3.8 getEpistemicClassForGlobalState()	78
5.44.3.9 isAgentInCoalition()	78
5.44.3.10 isGlobalTransitionControlledByCoalition()	79
5.44.3.11 newHistoryMarkDecisionAsInvalid()	79
5.44.3.12 printCurrentHistory()	80
5.44.3.13 restoreHistory()	80
5.44.3.14 revertLastDecision()	80
5.44.3.15 undoHistoryUntil()	81
5.44.3.16 undoLastHistoryEntry()	81
5.44.3.17 verify()	81
5.44.3.18 verifyGlobalState()	82

5.44.3.19 verifyLocalStates()	82
5.44.3.20 verifyTransitionSets()	82
5.45 yy_buffer_state Struct Reference	83
5.45.1 Member Data Documentation	83
5.45.1.1 yy_bs_column	83
5.45.1.2 yy_bs_lineno	84
5.46 yy_trans_info Struct Reference	84
5.47 yyalloc Union Reference	84
5.48 YYSTYPE Union Reference	84
6 File Documentation	85
6.1 Agent.cpp File Reference	85
6.1.1 Detailed Description	85
6.2 Agent.hpp File Reference	85
6.2.1 Detailed Description	85
6.3 Common.hpp File Reference	86
6.3.1 Detailed Description	86
6.4 DotGraph.cpp File Reference	86
6.4.1 Detailed Description	86
6.5 DotGraph.hpp File Reference	86
6.5.1 Detailed Description	87
6.5.2 Enumeration Type Documentation	87
6.5.2.1 DotGraphBase	87
6.6 EpistemicClass.hpp File Reference	87
6.6.1 Detailed Description	87
6.7 expressions.cc File Reference	88
6.7.1 Detailed Description	88
6.8 expressions.hpp File Reference	88
6.8.1 Detailed Description	89
6.9 GlobalModel.hpp File Reference	89
6.9.1 Detailed Description	89
6.10 GlobalModelGenerator.cpp File Reference	90
6.10.1 Detailed Description	90
6.11 GlobalModelGenerator.hpp File Reference	90
6.11.1 Detailed Description	90
6.12 GlobalState.cpp File Reference	91
6.12.1 Detailed Description	91
6.13 GlobalState.hpp File Reference	91
6.13.1 Detailed Description	91
6.14 GlobalTransition.cpp File Reference	91
6.14.1 Detailed Description	91
6.15 GlobalTransition.hpp File Reference	92

6.15.1 Detailed Description	92
6.16 LocalState.cpp File Reference	92
6.16.1 Detailed Description	92
6.17 LocalState.hpp File Reference	92
6.17.1 Detailed Description	92
6.18 ModelParser.cc File Reference	93
6.18.1 Detailed Description	93
6.19 nodes.cc File Reference	93
6.19.1 Detailed Description	93
6.20 nodes.hpp File Reference	94
6.20.1 Detailed Description	94
6.21 Types.hpp File Reference	94
6.21.1 Detailed Description	95
6.22 Utils.cpp File Reference	95
6.22.1 Detailed Description	96
6.22.2 Function Documentation	96
6.22.2.1 agentToString()	96
6.22.2.2 envToString()	96
6.22.2.3 getLocalStatesSCC()	97
6.22.2.4 localModelsToString()	97
6.22.2.5 outputGlobalModel()	97
6.22.2.6 tarjanVisit()	98
6.23 Utils.hpp File Reference	98
6.23.1 Function Documentation	99
6.23.1.1 agentToString()	99
6.23.1.2 envToString()	99
6.23.1.3 getLocalStatesSCC()	100
6.23.1.4 localModelsToString()	100
6.23.1.5 outputGlobalModel()	100
6.24 Verification.cpp File Reference	101
6.24.1 Detailed Description	101
6.24.2 Function Documentation	101
6.24.2.1 dbgHistEnt()	101
6.24.2.2 dbgVerifStatus()	102
6.24.2.3 verStatusToStr()	102
6.25 Verification.hpp File Reference	102
6.25.1 Enumeration Type Documentation	103
6.25.1.1 HistoryEntryType	103
6.25.1.2 TraversalMode	103
6.25.2 Function Documentation	104
6.25.2.1 verStatusToStr()	104

Chapter 1

STV2 - StraTegic Verifier 2

1.1 Usage

To run:

```
cd build
make clean
make
./stv
```

or

```
cd build
./build-run
```

Configuration file:

```
build/config.txt
```

CLI configuration overwrite:

```
# Input model
./stv --file PATH_TO_MODEL
./stv -f PATH_TO_MODEL
# Mode
./stv -m 0 #
./stv -m 1 # generate GlobalModel
./stv -m 2 # run verification
./stv -m 3 # same as 1 && 2
# Flags
# --OUTPUT_GLOBAL_MODEL      stdout data on global model (after expandAllStates)
# --OUTPUT_LOCAL_MODELS     stdout data on local models ()
# --OUTPUT_DOT_FILES        generate .dot files for agent templates, local and global models
# --ADD_EPSILON_TRANSITIONS generate global models with epsilon transitions
# --OVERWRITE_FORMULA       replace the formula from the model file with a different one
# --COUNTEREXAMPLE          output counterexample path if the formula verification returns an ERR
# --REDUCE                  reduce the amount of states and transitions using a DFS-POR algorithm and
                           select the first correct transition
# --REDUCE_ALL              reduce the amount of states and transitions using a DFS-POR algorithm and
                           select all available transitions
```

1.2 Tests

To run tests:

```
cd build
make clean
make sample_test
./sample_test
```

or

```
cd build
```

```
./build-test
```

To run larger tests:

```
cd build
make clean
make sample_test
./sample_test
```

or

```
cd build
./build-big-test
```

You might need to run

```
ulimit -s unlimited
```

beforehand

1.3 Performance estimation

Ubuntu/WSL:

```
# Minimal
> /usr/bin/time -f "%M\t%e" ./stv
# %M - maximum resident set size in KB
# %e - elapsed real time (wall clock) in seconds
# More detailed
> /usr/bin/time -f "time result\ncmd:%C\nreal %es\nuser %Us\nsys %Ss\nmemory:%MKB\ncpu %P" ./stv
# %C command line and arguments
# %e elapsed real time (wall clock) in seconds
# %U user time in seconds
# %S system (kernel) time in seconds
# %M maximum resident set size in KB
# %P percent of CPU this job got
# Full (verbose)
> /usr/bin/time -v ./stv
```

1.4 Specification

The specification language was inspired by ISPL (Interpreted Systems Programming Language) from [MCMAS](#).

The detailed syntax for the input format can be derived from `*./src/reader/{parser.y,scanner.l}*`, which intrinsically make up an EBNF grammar.

For the most parts, it is simple enough to get intuition just from looking at example's source code and the program's output.

IMPORTANT NOTES:

1. the (local) action names must be unique;
2. the transition relation (from the global model) should be serial;

1.5 Examples and templates

In `*/examples*` and `*/tests/examples*` there are several ready-to-use MAS specification files together with a proposed property (captured by ATL formula) for verification.

Often, we would want to reason about different (data-)configurations of the same system.

Using the templates we can parameterize the system specification, such that we only need to describe its dynamic behaviour.

A template can be fed with a configuration data to generate a concrete instance of a system.

Moreover, their use is independent from the tool: one can choose any templating engine (of the myriads available) or even write a custom one from the scratch.

Here, we utilize the [EJS](#) templating engine.

It has a CLI support, which is comes in handy for the tests/benchmarks that involve systems in multiple configurations.

```
# EJS feeds the data (as a list of key:val pairs) to the template file to generate the output:
> npm exec -- ejs TEMPLATE_FILE.ejs -i "{PARAM1:VAL1,PARAM2:VAL2,...}" -o OUTPUT_FILE.txt
# Possible generation query for the "trains":
> npm exec -- ejs Trains.ejs -i '{"N_TRAINS":3,"WITH_FORMULA":1}' -o 3Trains1Controller.txt
# Possible generation query for the "simple voting":
> npm exec -- ejs Simple_voting.ejs -i '{"N_VOTERS":2,"N_CANDIDATES":1,"WITH_FORMULA":0}' -o
    2Voters1Coercer1Candidate.txt
```

1.6 Misc

With `OUTPUT_DOT_FILES` flag the program outputs `*.dot*` files for templates, local and global models where:

- nodes are labelled with its location name (comma-separated for the global state)
- shared transitions are denoted by blue colour

Use Graphviz ([link](#)) to view in other format (eps, pdf, jpeg, etc.):

```
# Analogously for other formats
dot -Tpng lts_of_AGENT.dot > lts_of_AGENT.png
```

For the smaller graphs use `dot2png.sh` script, which converts all `*.dot*` files from a current folder to `*.png*`.

For bigger ones use `svg` format (may be viewed in Inkscape) and `dot2svg.sh`.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Agent	11
AgentTemplate	12
Assignment	17
Cfg	18
Condition	19
DotGraph	20
EpistemicClass	22
ExprNode	45
ExprAdd	22
ExprAnd	24
ExprConst	25
ExprDiv	26
ExprEq	28
ExprGe	29
ExprGt	31
ExprHart	33
ExprIdent	34
ExprKnow	37
ExprLe	38
ExprLt	40
ExprMul	42
ExprNe	43
ExprNot	46
ExprOr	47
ExprRem	48
ExprSub	50
Formula	52
FormulaTemplate	52
GlobalModel	53
GlobalModelGenerator	53
GlobalState	61
GlobalModelGenerator::GlobalStateTupleHash	62
GlobalTransition	63
HistoryDbg	63
HistoryEntry	65

LocalModels	67
LocalState	67
LocalStateTemplate	69
LocalTransition	69
ModelParser	70
TestVerif	71
TransitionTemplate	71
Var	73
Verification	73
yy_buffer_state	83
yy_trans_info	84
yyalloc	84
YYSTYPE	84

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Agent	Contains all data for a single Agent , including id, name and all of the agents' variables	11
AgentTemplate	Represents a single agent loaded from the description from a file	12
Assignment	Represents an assingment	17
Cfg	18
Condition	Represents a condition for LocalTransition	19
DotGraph	20
EpistemicClass	Represents a single epistemic class	22
ExprAdd	Node for addition	22
ExprAnd	Node for AND operator	24
ExprConst	Node for a constant	25
ExprDiv	Node for division	26
ExprEq	Node for "==" operator	28
ExprGe	Node for ">=" operator	29
ExprGt	Node for ">" operator	31
ExprHart	Node for a Hartley operator	33
ExprIdent	Node for an identifier	34
ExprKnow	Node for a knowledge operator	37
ExprLe	Node for "<=" operator	38
ExprLt	Node for "<" operator	40

ExprMul	Node for multiplication	42
ExprNe	Node for "!=" operator	43
ExprNode	Base node for expressions	45
ExprNot	Node for NOT operator	46
ExprOr	Node for OR operator	47
ExprRem	Node for modulo	48
ExprSub	Node for subtraction	50
Formula	52
FormulaTemplate	Contains a template for coalition of Agent as string from the formula	52
GlobalModel	Represents a global model, containing agents and a formula	53
GlobalModelGenerator	Stores the local models, formula and a global model	53
GlobalState	Represents a single global state	61
GlobalModelGenerator::GlobalStateTupleHash	62
GlobalTransition	Represents a single global transition	63
HistoryDbg	Stores history and allows displaying it to the console	63
HistoryEntry	Structure used to save model traversal history	65
LocalModels	Represents a single local model, contains all agents and variables	67
LocalState	Represents a single LocalState , containing id, name and internal variables	67
LocalStateTemplate	A template for the local state	69
LocalTransition	Represents a single local transition, containing id, global name, local name, is shared and count of the appearances	69
ModelParser	A parser for converting a text file into a model	70
TestVerif	71
TransitionTemplate	Represents a meta-transition	71
Var	Represents a variable in the model, containing name, initial value and persistence	73
Verification	A class that verifies if the model fulfills the formula. Also can do some operations on decision history	73
yy_buffer_state	83
yy_trans_info	84
yyalloc	84
YYSTYPE	84

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Agent.cpp	Class of an agent. Class of an agent	85
Agent.hpp	Class of an agent. Class of an agent	85
clock_test.h		??
Common.hpp	Contains all commonly used classes and structs. Contains all commonly used classes and structs	86
ConditionOperator.hpp		??
config.h		??
Constants.hpp		??
ctl_test.h		??
DotGraph.cpp	Class for drawing graphs of input models. Class used for making graph files of input models created by parsing the data and adding graphviz syntax	86
DotGraph.hpp	Class for drawing graphs of input models. Class used for making graph files of input models created by parsing the data and adding graphviz syntax	86
EpistemicClass.hpp	Struct of an epistemic class. Struct of an epistemic class	87
expressions.cc	Eval and helper class for expressions. Eval and helper class for expressions	88
expressions.hpp	Eval and helper class for expressions. Eval and helper class for expressions	88
general_test.h		??
GlobalModel.hpp	Struct of a global model. Struct of a global model	89
GlobalModelGenerator.cpp	Generator of a global model. Class for initializing and generating a global model	90
GlobalModelGenerator.hpp	Generator of a global model. Class for initializing and generating a global model	90
GlobalState.cpp	Struct representing a global state. Struct representing a global state	91
GlobalState.hpp	Struct representing a global state. Struct representing a global state	91

GlobalStateVerificationStatus.hpp	??
GlobalTransition.cpp	
Struct representing a global transition. Struct representing a global transition	91
GlobalTransition.hpp	
Struct representing a global transition. Struct representing a global transition	92
hartley_test.h	??
input_error_test.h	??
knowledge_test.h	??
LocalState.cpp	
Class representing a local state. Class representing a local state	92
LocalState.hpp	
Class representing a local state. Class representing a local state	92
LocalTransition.hpp	??
ModelParser.cc	
A model parser. A parser for converting a text file into a model	93
ModelParser.hpp	??
nodes.cc	
Parser templates. Class for setting up a new objects from a parser	93
nodes.hpp	
Parser templates. Class for setting up a new objects from a parser	94
parser.h	??
recursion_test.h	??
reduction_test.h	??
simple_voting_run_test.h	??
simple_voting_run_test_but_faster.h	??
simple_voting_test.h	??
simple_voting_test_but_faster.h	??
simple_voting_with_fakes_test.h	??
test_test.h	??
trains_test.h	??
trains_with_bridge_test.h	??
tree_test.h	??
Types.hpp	
Custom data structures. Data structures and classes containing model data	94
Utils.cpp	
Utility functions. A collection of utility functions to use in the project	95
Utils.hpp	98
Verification.cpp	
Class for verification of the formula on a model. Class for verification of the specified formula on a specified model	101
Verification.hpp	102

Chapter 5

Class Documentation

5.1 Agent Class Reference

Contains all data for a single [Agent](#), including id, name and all of the agents' variables.

```
#include <Agent.hpp>
```

Public Member Functions

- [Agent](#) (int _id, string _name)
Constructor for the [Agent](#) class, assigning it an id and name.
- [LocalState](#) * [includesState](#) ([LocalState](#) *state)
Checks if there is an equivalent [LocalState](#) in the model to the one passed as an argument.

Public Attributes

- int [id](#)
Identifier of the agent.
- string [name](#)
Name of the agent.
- set< [Var](#) * > [vars](#)
Variable names for the agent.
- [LocalState](#) * [initState](#)
Initial state of the agent.
- vector< [LocalState](#) * > [localStates](#)
Local states for this agent.
- vector< [LocalTransition](#) * > [localTransitions](#)
Local transitions for this agent.

5.1.1 Detailed Description

Contains all data for a single [Agent](#), including id, name and all of the agents' variables.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Agent()

```
Agent::Agent (
    int _id,
    string _name ) [inline]
```

Constructor for the [Agent](#) class, assigning it an id and name.

Parameters

<code>_id</code>	Identifier of the new agent.
<code>_name</code>	Name of the new agent.

5.1.3 Member Function Documentation

5.1.3.1 includesState()

```
LocalState * Agent::includesState (
    LocalState * state )
```

Checks if there is an equivalent [LocalState](#) in the model to the one passed as an argument.

Parameters

<code>state</code>	A pointer to LocalState to be checked.
--------------------	--

Returns

Returns a pointer to an equivalent [LocalState](#) if such exists, otherwise returns NULL.

The documentation for this class was generated from the following files:

- [Agent.hpp](#)
- [Agent.cpp](#)

5.2 AgentTemplate Class Reference

Represents a single agent loaded from the description from a file.

```
#include <nodes.hpp>
```


Public Member Functions

- [AgentTemplate](#) ()
Constructor for an [AgentTemplate](#).
- virtual [AgentTemplate](#) & [setIdent](#) (string _ident)
Set the identifier of an agent.
- virtual [AgentTemplate](#) & [setInitState](#) (string _startState)
Set the initial state of the agent.
- virtual [AgentTemplate](#) & [addLocal](#) (set< string > *variables)
Adds local variables to an agent.
- virtual [AgentTemplate](#) & [addPersistent](#) (set< string > *variables)
Adds persistent variables to an agent.
- virtual [AgentTemplate](#) & [addInitial](#) (set< [Assignment](#) * > *assigns)
Adds initial assignments.
- virtual [AgentTemplate](#) & [addTransition](#) ([TransitionTemplate](#) *_transition)
Adds a transition to the agent.
- virtual [Agent](#) * [generateAgent](#) (int id)
Generate a new agent for the model.

Friends

- class [DotGraph](#)

5.2.1 Detailed Description

Represents a single agent loaded from the description from a file.

5.2.2 Member Function Documentation

5.2.2.1 addInitial()

```
AgentTemplate & AgentTemplate::addInitial (
    set< Assignment * > * assigns ) [virtual]
```

Adds initial assignments.

Sets initial values of agent's variables.

Parameters

<i>assigns</i>	Assignments to be added.
----------------	--------------------------

Returns

Returns a pointer to self.

Parameters

<i>assigns</i>	Set of variables to assign.
----------------	-----------------------------

Returns

Returns itself.

5.2.2.2 addLocal()

```
AgentTemplate & AgentTemplate::addLocal (
    set< string > * variables ) [virtual]
```

Adds local variables to an agent.

Adds local variables to the agent.

Parameters

<i>variables</i>	Set of variables to be added.
------------------	-------------------------------

Returns

Returns a pointer to self.

Parameters

<i>variables</i>	A pointer to a set of strings with the variables to be added.
------------------	---

Returns

Returns itself.

5.2.2.3 addPersistent()

```
AgentTemplate & AgentTemplate::addPersistent (
    set< string > * variables ) [virtual]
```

Adds persistent variables to an agent.

Adds persistent variables to the agent.

Parameters

<i>variables</i>	Set of variables to be added.
------------------	-------------------------------

Returns

Returns a pointer to self.

Parameters

<i>variables</i>	A pointer to a set of strings with the variables to be added.
------------------	---

Returns

Returns itself.

5.2.2.4 addTransition()

```
AgentTemplate & AgentTemplate::addTransition (
    TransitionTemplate * _transition ) [virtual]
```

Adds a transition to the agent.

Adds a transition for the agent.

Parameters

<i>_transition</i>	Transition to be added.
--------------------	-------------------------

Returns

Returns a pointer to self.

Parameters

<i>_transition</i>	Transition to be added.
--------------------	-------------------------

Returns

Returns itself.

5.2.2.5 generateAgent()

```
Agent * AgentTemplate::generateAgent (
    int id ) [virtual]
```

Generate a new agent for the model.

Generates an agent for the model.

Parameters

<i>id</i>	Identification number defining a new Agent .
-----------	--

Returns

Returns a pointer to a new [Agent](#).

Parameters

<i>id</i>	Identifier of the new Agent .
-----------	---

Returns

Returns a pointer to a newly created [Agent](#).

5.2.2.6 setIdent()

```
AgentTemplate & AgentTemplate::setIdent (
    string _ident ) [virtual]
```

Set the identifier of an agent.

Sets the identifier of an agent.

Parameters

<i>_ident</i>	New agent identifier.
---------------	-----------------------

Returns

Returns a pointer to self.

Parameters

<i>_ident</i>	String with a new identifier.
---------------	-------------------------------

Returns

Returns itself.

5.2.2.7 setInitState()

```
AgentTemplate & AgentTemplate::setInitState (
    string _initState ) [virtual]
```

Set the initial state of the agent.

Sets initial state of an agent.

Parameters

<code>_startState</code>	New initial agent state.
--------------------------	--------------------------

Returns

Returns a pointer to self.

Parameters

<code>_initState</code>	String with a new state.
-------------------------	--------------------------

Returns

Returns itself.

The documentation for this class was generated from the following files:

- [nodes.hpp](#)
- [nodes.cc](#)

5.3 Assignment Class Reference

Represents an assingment.

```
#include <nodes.hpp>
```

Public Member Functions

- [Assignment](#) (string _ident, [ExprNode](#) *_exp)
Constructor for an [Assignment](#) class.
- virtual void [assign](#) ([Environment](#) &env)
Make an assignment in a given environment.

Public Attributes

- string [ident](#)
To what we should assign a value.
- [ExprNode](#) * [value](#)
A value to be assigned.

5.3.1 Detailed Description

Represents an assingment.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 Assignment()

```
Assignment::Assignment (
    string _ident,
    ExprNode * _exp ) [inline]
```

Constructor for an [Assignment](#) class.

Parameters

<code>_ident</code>	To what we should assign a value.
<code>_exp</code>	A value to be assigned.

5.3.3 Member Function Documentation

5.3.3.1 assign()

```
virtual void Assignment::assign (
    Environment & env ) [inline], [virtual]
```

Make an assignment in a given environment.

Parameters

<code>env</code>	Environment in which to make an assignment.
------------------	---

The documentation for this class was generated from the following file:

- [nodes.hpp](#)

5.4 Cfg Struct Reference

Public Attributes

- `std::string` [fname](#)

- path to input file with system specification*
- int [stv_mode](#)
 - stv_code as sum/combination of (1 - expandAllStates, 2 - verify, 4 - print metadata, 8 - run experiments)*
- bool [output_local_models](#)
 - (obsolete) print data on local model*
- bool [output_global_model](#)
 - (obsolete) print data on local model*
- bool [output_dot_files](#)
 - flag for .dot export (by default exports templates and local/global models)*
- std::string [dotdir](#)
 - pathprefix for .dot files export*
- int **model_id**
- bool [add_epsilon_transitions](#)
 - add epsilon transitions to the states in the model when it's blocked for some reason*
- bool **formula_from_parameter**
- std::string **formula**
- bool **counterexample**
- bool **reduce**
- bool **reduce_all**
- std::string **reduce_args**

The documentation for this struct was generated from the following file:

- [Common.hpp](#)

5.5 Condition Struct Reference

Represents a condition for [LocalTransition](#).

```
#include <Types.hpp>
```

Public Attributes

- [Var * var](#)
 - Pointer to a variable.*
- ConditionOperator [conditionOperator](#)
 - Conditional operator for the variable.*
- int [comparedValue](#)
 - Condition value to be met.*

5.5.1 Detailed Description

Represents a condition for [LocalTransition](#).

The documentation for this struct was generated from the following file:

- [Types.hpp](#)

5.6 DotGraph Class Reference

Public Member Functions

- [DotGraph](#) ()
empty graph
- [DotGraph](#) ([GlobalModel](#) *const gm, bool extended=false, bool correct=false)
parses the transitions/states templates into nodes/edges
- [DotGraph](#) ([Agent](#) *const ag, bool extended=false)
parses the local transitions/states templates into nodes/edges
- [DotGraph](#) ([AgentTemplate](#) *const at)
parses the edge/state templates into nodes/edges
- void [saveToFile](#) (std::string pathprefix, std::string nameprefix, std::string basename="")
creates a .dot file

Public Attributes

- std::vector< std::string > **nodes**
- std::vector< std::string > **edges**
- std::string **graphName**
- [DotGraphBase](#) **graphBase**

Static Public Attributes

- static string [styleString](#)
(temporary hard-coded) graphviz visual configuration

Protected Member Functions

- void [addNode](#) (std::string id, std::string name)
creates a node string in graphviz syntax
- void [addEdge](#) (std::string src, std::string trg, std::string label)
creates an edge string in graphviz syntax

5.6.1 Member Function Documentation

5.6.1.1 addEdge()

```
void DotGraph::addEdge (
    std::string src,
    std::string trg,
    std::string label ) [protected]
```

creates an edge string in graphviz syntax

Parameters

<i>src</i>	id of a source node
<i>trg</i>	id of a target node
<i>label</i>	edge label and, possibly, extra attributes

5.6.1.2 addNode()

```
void DotGraph::addNode (
    std::string id,
    std::string name ) [protected]
```

creates a node string in graphviz syntax

Parameters

<i>id</i>	unique internal node identifier
<i>name</i>	displayed node label/name

5.6.1.3 saveToFile()

```
void DotGraph::saveToFile (
    std::string pathprefix,
    std::string nameprefix,
    std::string basename = "" )
```

creates a .dot file

Parameters

<i>basename</i>	name of file (parent graph name if blank)
-----------------	---

5.6.2 Member Data Documentation**5.6.2.1 styleString**

```
std::string DotGraph::styleString [static]
```

Initial value:

=

```

"\tedge[fontsize=\"10\"]\n"
"\tnode [\n"
"\t\tshape=circle,\n"

"\t\twidth=auto,\n"
"\t\tcolor=\"black\",\n"
"\t\tfillcolor=\"#eeeeee\",\n"
"\t\tstyle=\"filled,solid\",\n"
"\t\tfontsize=8,\n"
"\t\tfontname=\"Roboto\"\n"
"\t]\n"
"\tfontname=Consolas\n"
"\tlayout=dot\n"

```

(temporary hard-coded) graphviz visual configuration

The documentation for this class was generated from the following files:

- [DotGraph.hpp](#)
- [DotGraph.cpp](#)

5.7 EpistemicClass Struct Reference

Represents a single epistemic class.

```
#include <EpistemicClass.hpp>
```

Public Attributes

- string [hash](#)
Hash of that epistemic class.
- map< string, [GlobalState](#) * > [globalStates](#)
Map of [GlobalState](#) hashes to according [GlobalState](#) pointers bound to this epistemic class.
- [GlobalTransition](#) * [fixedCoalitionTransition](#)
Transition that was already selected in this epistemic class. Model has to choose this transition if it is already set.

5.7.1 Detailed Description

Represents a single epistemic class.

The documentation for this struct was generated from the following file:

- [EpistemicClass.hpp](#)

5.8 ExprAdd Class Reference

Node for addition.

```
#include <expressions.hpp>
```

Public Member Functions

- [ExprAdd](#) ([ExprNode](#) * _larg, [ExprNode](#) * _rarg)
Addition expression constructor.
- virtual int [eval](#) ([Environment](#) &env, [GlobalModelGenerator](#) *generator, [GlobalState](#) *globalState)
Calculates the expression value.
- virtual int [eval](#) ([Environment](#) &env)

5.8.1 Detailed Description

Node for addition.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 ExprAdd()

```
ExprAdd::ExprAdd (
    ExprNode * _larg,
    ExprNode * _rarg ) [inline]
```

Addition expression constructor.

Parameters

<i>_larg</i>	Left argument of the expression.
<i>_rarg</i>	Right argument of the expression.

5.8.3 Member Function Documentation

5.8.3.1 eval()

```
int ExprAdd::eval (
    Environment & env,
    GlobalModelGenerator * generator,
    GlobalState * globalState ) [virtual]
```

Calculates the expression value.

Parameters

<i>env</i>	Environment values.
------------	---------------------

Returns

Returns an integer.

Implements [ExprNode](#).

The documentation for this class was generated from the following files:

- [expressions.hpp](#)
- [expressions.cc](#)

5.9 ExprAnd Class Reference

Node for AND operator.

```
#include <expressions.hpp>
```

Public Member Functions

- [ExprAnd](#) ([ExprNode](#) *_larg, [ExprNode](#) *_rarg)
Logic AND expression constructor.
- virtual int **eval** ([Environment](#) &env, [GlobalModelGenerator](#) *generator, [GlobalState](#) *globalState)
Calculates the expression value.
- virtual int **eval** ([Environment](#) &env)

5.9.1 Detailed Description

Node for AND operator.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 ExprAnd()

```
ExprAnd::ExprAnd (
    ExprNode * _larg,
    ExprNode * _rarg ) [inline]
```

Logic AND expression constructor.

Parameters

<i>_larg</i>	Left argument of the expression.
<i>_rarg</i>	Right argument of the expression.

5.9.3 Member Function Documentation

5.9.3.1 eval()

```
int ExprAnd::eval (
    Environment & env,
    GlobalModelGenerator * generator,
    GlobalState * globalState ) [virtual]
```

Calculates the expression value.

Parameters

<i>env</i>	Environment values.
------------	---------------------

Returns

Returns an integer.

Implements [ExprNode](#).

The documentation for this class was generated from the following files:

- [expressions.hpp](#)
- [expressions.cc](#)

5.10 ExprConst Class Reference

Node for a constant.

```
#include <expressions.hpp>
```

Public Member Functions

- [ExprConst](#) (int _val)
Constant expression constructor.
- virtual int [eval](#) ([Environment](#) &env, [GlobalModelGenerator](#) *generator, [GlobalState](#) *globalState)
Calculates the expression value.
- virtual int **eval** ([Environment](#) &env)

5.10.1 Detailed Description

Node for a constant.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 ExprConst()

```
ExprConst::ExprConst (
    int _val ) [inline]
```

Constant expression constructor.

Parameters

<code>_val</code>	ExprConst value.
-------------------	------------------

5.10.3 Member Function Documentation

5.10.3.1 eval()

```
int ExprConst::eval (
    Environment & env,
    GlobalModelGenerator * generator,
    GlobalState * globalState ) [virtual]
```

Calculates the expression value.

Parameters

<code>env</code>	Environment values.
------------------	---------------------

Returns

Returns an integer.

Implements [ExprNode](#).

The documentation for this class was generated from the following files:

- [expressions.hpp](#)
- [expressions.cc](#)

5.11 ExprDiv Class Reference

Node for division.

```
#include <expressions.hpp>
```

Public Member Functions

- [ExprDiv](#) ([ExprNode](#) **_larg*, [ExprNode](#) **_rarg*)
Division expression constructor.
- virtual int [eval](#) ([Environment](#) &*env*, [GlobalModelGenerator](#) **generator*, [GlobalState](#) **globalState*)
Calculates the expression value.
- virtual int [eval](#) ([Environment](#) &*env*)

5.11.1 Detailed Description

Node for division.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 ExprDiv()

```
ExprDiv::ExprDiv (
    ExprNode * _larg,
    ExprNode * _rarg ) [inline]
```

Division expression constructor.

Parameters

<i>_larg</i>	Left argument of the expression.
<i>_rarg</i>	Right argument of the expression.

5.11.3 Member Function Documentation

5.11.3.1 eval()

```
int ExprDiv::eval (
    Environment & env,
    GlobalModelGenerator * generator,
    GlobalState * globalState ) [virtual]
```

Calculates the expression value.

Parameters

<i>env</i>	Environment values.
------------	---------------------

Returns

Returns an integer.

Implements [ExprNode](#).

The documentation for this class was generated from the following files:

- [expressions.hpp](#)
- [expressions.cc](#)

5.12 ExprEq Class Reference

Node for "==" operator.

```
#include <expressions.hpp>
```

Public Member Functions

- [ExprEq](#) ([ExprNode](#) *_larg, [ExprNode](#) *_rarg)
Equals expression constructor.
- virtual int [eval](#) ([Environment](#) &env, [GlobalModelGenerator](#) *generator, [GlobalState](#) *globalState)
Calculates the expression value.
- virtual int **eval** ([Environment](#) &env)

5.12.1 Detailed Description

Node for "==" operator.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 ExprEq()

```
ExprEq::ExprEq (
    ExprNode * _larg,
    ExprNode * _rarg ) [inline]
```

Equals expression constructor.

Parameters

_larg	Left argument of the expression.
_rarg	Right argument of the expression.

5.12.3 Member Function Documentation

5.12.3.1 eval()

```
int ExprEq::eval (
    Environment & env,
    GlobalModelGenerator * generator,
    GlobalState * globalState ) [virtual]
```

Calculates the expression value.

Parameters

<i>env</i>	Environment values.
------------	---------------------

Returns

Returns an integer.

Implements [ExprNode](#).

The documentation for this class was generated from the following files:

- [expressions.hpp](#)
- [expressions.cc](#)

5.13 ExprGe Class Reference

Node for ">=" operator.

```
#include <expressions.hpp>
```

Public Member Functions

- [ExprGe](#) ([ExprNode](#) *_larg, [ExprNode](#) *_rarg)
Greater or equal expression constructor.
- virtual int [eval](#) ([Environment](#) &env, [GlobalModelGenerator](#) *generator, [GlobalState](#) *globalState)
Calculates the expression value.
- virtual int **eval** ([Environment](#) &env)

5.13.1 Detailed Description

Node for ">=" operator.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 ExprGe()

```
ExprGe::ExprGe (
    ExprNode * _larg,
    ExprNode * _rarg ) [inline]
```

Greater or equal expression constructor.

Parameters

<i>_larg</i>	Left argument of the expression.
<i>_rarg</i>	Right argument of the expression.

5.13.3 Member Function Documentation

5.13.3.1 eval()

```
int ExprGe::eval (
    Environment & env,
    GlobalModelGenerator * generator,
    GlobalState * globalState ) [virtual]
```

Calculates the expression value.

Parameters

<i>env</i>	Environment values.
------------	---------------------

Returns

Returns an integer.

Implements [ExprNode](#).

The documentation for this class was generated from the following files:

- [expressions.hpp](#)
- [expressions.cc](#)

5.14 ExprGt Class Reference

Node for ">" operator.

```
#include <expressions.hpp>
```

Public Member Functions

- [ExprGt](#) ([ExprNode](#) *_larg, [ExprNode](#) *_rarg)
Greater than expression constructor.
- virtual int [eval](#) ([Environment](#) &env, [GlobalModelGenerator](#) *generator, [GlobalState](#) *globalState)
Calculates the expression value.
- virtual int [eval](#) ([Environment](#) &env)

5.14.1 Detailed Description

Node for ">" operator.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 ExprGt()

```
ExprGt::ExprGt (
    ExprNode * _larg,
    ExprNode * _rarg ) [inline]
```

Greater than expression constructor.

Parameters

_larg	Left argument of the expression.
_rarg	Right argument of the expression.

5.14.3 Member Function Documentation

5.14.3.1 eval()

```
int ExprGt::eval (
    Environment & env,
```

```
GlobalModelGenerator * generator,  
GlobalState * globalState ) [virtual]
```

Calculates the expression value.

Parameters

<i>env</i>	Environment values.
------------	---------------------

Returns

Returns an integer.

Implements [ExprNode](#).

The documentation for this class was generated from the following files:

- [expressions.hpp](#)
- [expressions.cc](#)

5.15 ExprHart Class Reference

Node for a Hartley operator.

```
#include <expressions.hpp>
```

Public Member Functions

- [ExprHart](#) (string *_agentName*, bool *_le*, int *_val*, vector< [ExprNode](#) * > **_arg*)
Constant expression constructor.
- virtual int [eval](#) ([Environment](#) &*env*, [GlobalModelGenerator](#) **generator*, [GlobalState](#) **globalState*)
Calculates the expression value.
- virtual int **eval** ([Environment](#) &*env*)

5.15.1 Detailed Description

Node for a Hartley operator.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 ExprHart()

```
ExprHart::ExprHart (
    string _agentName,
    bool _le,
    int _val,
    vector< ExprNode * > *_arg ) [inline]
```

Constant expression constructor.

Parameters

<code>_agentName</code>	Agent name for Hartley operator.
<code>_le</code>	Flag for if Hartley coefficient should be less equal or greater equal.
<code>_val</code>	Number to compare the result to.
<code>_arg</code>	Expression to verify with the given knowledge.

5.15.3 Member Function Documentation

5.15.3.1 `eval()`

```
int ExprHart::eval (
    Environment & env,
    GlobalModelGenerator * generator,
    GlobalState * globalState ) [virtual]
```

Calculates the expression value.

Parameters

<code>env</code>	Environment values.
------------------	---------------------

Returns

Returns an integer.

Implements [ExprNode](#).

The documentation for this class was generated from the following files:

- [expressions.hpp](#)
- [expressions.cc](#)

5.16 ExprIdent Class Reference

Node for an identifier.

```
#include <expressions.hpp>
```

Public Member Functions

- [ExprIdent](#) (string _ident)
Identifier expression constructor.
- virtual int [eval](#) ([Environment](#) &env, [GlobalModelGenerator](#) *generator, [GlobalState](#) *globalState)
Calculates the expression value.
- virtual int [eval](#) ([Environment](#) &env)

5.16.1 Detailed Description

Node for an identifier.

5.16.2 Constructor & Destructor Documentation

5.16.2.1 ExprIdent()

```
ExprIdent::ExprIdent (
    string _ident ) [inline]
```

Identifier expression constructor.

Parameters

<code>_ident</code>	ExprIdent value.
---------------------	------------------

5.16.3 Member Function Documentation

5.16.3.1 eval() [1/2]

```
int ExprIdent::eval (
    Environment & env ) [virtual]
```

Parameters

<code>env</code>	
------------------	--

Returns

Implements [ExprNode](#).

5.16.3.2 eval() [2/2]

```
int ExprIdent::eval (
    Environment & env,
```

```
GlobalModelGenerator * generator,  
GlobalState * globalState ) [virtual]
```

Calculates the expression value.

Parameters

<i>env</i>	Environment values.
------------	---------------------

Returns

Returns an integer.

Implements [ExprNode](#).

The documentation for this class was generated from the following files:

- [expressions.hpp](#)
- [expressions.cc](#)

5.17 ExprKnow Class Reference

Node for a knowledge operator.

```
#include <expressions.hpp>
```

Public Member Functions

- [ExprKnow](#) (string *_agentName*, [ExprNode](#) **_arg*)
Constant expression constructor.
- virtual int [eval](#) ([Environment](#) &*env*, [GlobalModelGenerator](#) **generator*, [GlobalState](#) **globalState*)
Calculates the expression value.
- virtual int **eval** ([Environment](#) &*env*)

5.17.1 Detailed Description

Node for a knowledge operator.

5.17.2 Constructor & Destructor Documentation

5.17.2.1 ExprKnow()

```
ExprKnow::ExprKnow (
    string _agentName,
    ExprNode * _arg ) [inline]
```

Constant expression constructor.

Parameters

<code>_agentName</code>	Agent name for knowledge operator.
<code>_arg</code>	Expression to verify with the given knowledge.

5.17.3 Member Function Documentation

5.17.3.1 eval()

```
int ExprKnow::eval (
    Environment & env,
    GlobalModelGenerator * generator,
    GlobalState * globalState ) [virtual]
```

Calculates the expression value.

Parameters

<code>env</code>	Environment values.
------------------	---------------------

Returns

Returns an integer.

Implements [ExprNode](#).

The documentation for this class was generated from the following files:

- [expressions.hpp](#)
- [expressions.cc](#)

5.18 ExprLe Class Reference

Node for "<=" operator.

```
#include <expressions.hpp>
```

Public Member Functions

- [ExprLe](#) ([ExprNode](#) * _larg, [ExprNode](#) * _rarg)
Less or equal expression constructor.
- virtual int [eval](#) ([Environment](#) &env, [GlobalModelGenerator](#) *generator, [GlobalState](#) *globalState)
Calculates the expression value.
- virtual int **eval** ([Environment](#) &env)

5.18.1 Detailed Description

Node for "<=" operator.

5.18.2 Constructor & Destructor Documentation

5.18.2.1 ExprLe()

```
ExprLe::ExprLe (
    ExprNode * _larg,
    ExprNode * _rarg ) [inline]
```

Less or equal expression constructor.

Parameters

<i>_larg</i>	Left argument of the expression.
<i>_rarg</i>	Right argument of the expression.

5.18.3 Member Function Documentation

5.18.3.1 eval()

```
int ExprLe::eval (
    Environment & env,
    GlobalModelGenerator * generator,
    GlobalState * globalState ) [virtual]
```

Calculates the expression value.

Parameters

<i>env</i>	Environment values.
------------	---------------------

Returns

Returns an integer.

Implements [ExprNode](#).

The documentation for this class was generated from the following files:

- [expressions.hpp](#)
- [expressions.cc](#)

5.19 ExprLt Class Reference

Node for "<" operator.

```
#include <expressions.hpp>
```

Public Member Functions

- [ExprLt](#) ([ExprNode](#) *_larg, [ExprNode](#) *_rarg)
Less than expression constructor.
- virtual int [eval](#) ([Environment](#) &env, [GlobalModelGenerator](#) *generator, [GlobalState](#) *globalState)
Calculates the expression value.
- virtual int [eval](#) ([Environment](#) &env)

5.19.1 Detailed Description

Node for "<" operator.

5.19.2 Constructor & Destructor Documentation

5.19.2.1 ExprLt()

```
ExprLt::ExprLt (
    ExprNode * _larg,
    ExprNode * _rarg ) [inline]
```

Less than expression constructor.

Parameters

_larg	Left argument of the expression.
_rarg	Right argument of the expression.

5.19.3 Member Function Documentation

5.19.3.1 eval()

```
int ExprLt::eval (
    Environment & env,
```

```
GlobalModelGenerator * generator,  
GlobalState * globalState ) [virtual]
```

Calculates the expression value.

Parameters

<i>env</i>	Environment values.
------------	---------------------

Returns

Returns an integer.

Implements [ExprNode](#).

The documentation for this class was generated from the following files:

- [expressions.hpp](#)
- [expressions.cc](#)

5.20 ExprMul Class Reference

Node for multiplication.

```
#include <expressions.hpp>
```

Public Member Functions

- [ExprMul](#) ([ExprNode](#) *_larg, [ExprNode](#) *_rarg)
Multiplication expression constructor.
- virtual int [eval](#) ([Environment](#) &env, [GlobalModelGenerator](#) *generator, [GlobalState](#) *globalState)
Calculates the expression value.
- virtual int **eval** ([Environment](#) &env)

5.20.1 Detailed Description

Node for multiplication.

5.20.2 Constructor & Destructor Documentation

5.20.2.1 ExprMul()

```
ExprMul::ExprMul (  
    ExprNode * _larg,  
    ExprNode * _rarg ) [inline]
```

Multiplication expression constructor.

Parameters

<code>_larg</code>	Left argument of the expression.
<code>_rarg</code>	Right argument of the expression.

5.20.3 Member Function Documentation

5.20.3.1 eval()

```
int ExprMul::eval (
    Environment & env,
    GlobalModelGenerator * generator,
    GlobalState * globalState ) [virtual]
```

Calculates the expression value.

Parameters

<code>env</code>	Environment values.
------------------	---------------------

Returns

Returns an integer.

Implements [ExprNode](#).

The documentation for this class was generated from the following files:

- [expressions.hpp](#)
- [expressions.cc](#)

5.21 ExprNe Class Reference

Node for "!=" operator.

```
#include <expressions.hpp>
```

Public Member Functions

- [ExprNe](#) ([ExprNode](#) * _larg, [ExprNode](#) * _rarg)
Not equals expression constructor.
- virtual int [eval](#) ([Environment](#) &env, [GlobalModelGenerator](#) *generator, [GlobalState](#) *globalState)
Calculates the expression value.
- virtual int **eval** ([Environment](#) &env)

5.21.1 Detailed Description

Node for "!=" operator.

5.21.2 Constructor & Destructor Documentation

5.21.2.1 ExprNe()

```
ExprNe::ExprNe (
    ExprNode * _larg,
    ExprNode * _rarg ) [inline]
```

Not equals expression constructor.

Parameters

<i>_larg</i>	Left argument of the expression.
<i>_rarg</i>	Right argument of the expression.

5.21.3 Member Function Documentation

5.21.3.1 eval()

```
int ExprNe::eval (
    Environment & env,
    GlobalModelGenerator * generator,
    GlobalState * globalState ) [virtual]
```

Calculates the expression value.

Parameters

<i>env</i>	Environment values.
------------	---------------------

Returns

Returns an integer.

Implements [ExprNode](#).

The documentation for this class was generated from the following files:

- [expressions.hpp](#)
- [expressions.cc](#)

5.22 ExprNode Class Reference

Base node for expressions.

```
#include <expressions.hpp>
```

Public Member Functions

- virtual int [eval](#) ([Environment](#) &env, [GlobalModelGenerator](#) *generator, [GlobalState](#) *globalState)=0
Calculates the expression value.
- virtual int **eval** ([Environment](#) &env)=0

5.22.1 Detailed Description

Base node for expressions.

5.22.2 Member Function Documentation

5.22.2.1 eval()

```
virtual int ExprNode::eval (
    Environment & env,
    GlobalModelGenerator * generator,
    GlobalState * globalState ) [pure virtual]
```

Calculates the expression value.

Parameters

<i>env</i>	Environment values.
------------	---------------------

Returns

Returns an integer.

Implemented in [ExprHart](#), [ExprKnow](#), [ExprGe](#), [ExprGt](#), [ExprLe](#), [ExprLt](#), [ExprNe](#), [ExprEq](#), [ExprNot](#), [ExprOr](#), [ExprAnd](#), [ExprRem](#), [ExprDiv](#), [ExprMul](#), [ExprSub](#), [ExprAdd](#), [ExprIdent](#), and [ExprConst](#).

The documentation for this class was generated from the following file:

- [expressions.hpp](#)

5.23 ExprNot Class Reference

Node for NOT operator.

```
#include <expressions.hpp>
```

Public Member Functions

- [ExprNot](#) ([ExprNode](#) *_arg)
Logic NOT expression constructor.
- virtual int [eval](#) ([Environment](#) &env, [GlobalModelGenerator](#) *generator, [GlobalState](#) *globalState)
Calculates the expression value.
- virtual int [eval](#) ([Environment](#) &env)

5.23.1 Detailed Description

Node for NOT operator.

5.23.2 Constructor & Destructor Documentation

5.23.2.1 ExprNot()

```
ExprNot::ExprNot (
    ExprNode * _arg ) [inline]
```

Logic NOT expression constructor.

Parameters

_arg	Calculates the expression value.
----------------------	----------------------------------

5.23.3 Member Function Documentation

5.23.3.1 eval()

```
int ExprNot::eval (
    Environment & env,
    GlobalModelGenerator * generator,
    GlobalState * globalState ) [virtual]
```

Calculates the expression value.

Parameters

<i>env</i>	Environment values.
------------	---------------------

Returns

Returns an integer.

Implements [ExprNode](#).

The documentation for this class was generated from the following files:

- [expressions.hpp](#)
- [expressions.cc](#)

5.24 ExprOr Class Reference

Node for OR operator.

```
#include <expressions.hpp>
```

Public Member Functions

- [ExprOr](#) ([ExprNode](#) * _larg, [ExprNode](#) * _rarg)
Logic OR expression constructor.
- virtual int [eval](#) ([Environment](#) &env, [GlobalModelGenerator](#) *generator, [GlobalState](#) *globalState)
Calculates the expression value.
- virtual int **eval** ([Environment](#) &env)

5.24.1 Detailed Description

Node for OR operator.

5.24.2 Constructor & Destructor Documentation

5.24.2.1 ExprOr()

```
ExprOr::ExprOr (
    ExprNode * _larg,
    ExprNode * _rarg ) [inline]
```

Logic OR expression constructor.

Parameters

<code>_larg</code>	Left argument of the expression.
<code>_rarg</code>	Right argument of the expression.

5.24.3 Member Function Documentation

5.24.3.1 eval()

```
int ExprOr::eval (
    Environment & env,
    GlobalModelGenerator * generator,
    GlobalState * globalState ) [virtual]
```

Calculates the expression value.

Parameters

<code>env</code>	Environment values.
------------------	---------------------

Returns

Returns an integer.

Implements [ExprNode](#).

The documentation for this class was generated from the following files:

- [expressions.hpp](#)
- [expressions.cc](#)

5.25 ExprRem Class Reference

Node for modulo.

```
#include <expressions.hpp>
```

Public Member Functions

- [ExprRem](#) ([ExprNode](#) *_larg, [ExprNode](#) *_rarg)
Modulo expression constructor.
- virtual int [eval](#) ([Environment](#) &env, [GlobalModelGenerator](#) *generator, [GlobalState](#) *globalState)
Calculates the expression value.
- virtual int **eval** ([Environment](#) &env)

5.25.1 Detailed Description

Node for modulo.

5.25.2 Constructor & Destructor Documentation

5.25.2.1 ExprRem()

```
ExprRem::ExprRem (
    ExprNode * _larg,
    ExprNode * _rarg ) [inline]
```

Modulo expression constructor.

Parameters

<i>_larg</i>	Left argument of the expression.
<i>_rarg</i>	Right argument of the expression.

5.25.3 Member Function Documentation

5.25.3.1 eval()

```
int ExprRem::eval (
    Environment & env,
    GlobalModelGenerator * generator,
    GlobalState * globalState ) [virtual]
```

Calculates the expression value.

Parameters

<i>env</i>	Environment values.
------------	---------------------

Returns

Returns an integer.

Implements [ExprNode](#).

The documentation for this class was generated from the following files:

- [expressions.hpp](#)
- [expressions.cc](#)

5.26 ExprSub Class Reference

Node for subtraction.

```
#include <expressions.hpp>
```

Public Member Functions

- [ExprSub](#) ([ExprNode](#) *[_larg](#), [ExprNode](#) *[_rarg](#))
Subtraction expression constructor.
- virtual int [eval](#) ([Environment](#) &[env](#), [GlobalModelGenerator](#) *[generator](#), [GlobalState](#) *[globalState](#))
Calculates the expression value.
- virtual int [eval](#) ([Environment](#) &[env](#))

5.26.1 Detailed Description

Node for subtraction.

5.26.2 Constructor & Destructor Documentation

5.26.2.1 ExprSub()

```
ExprSub::ExprSub (
    ExprNode * \_larg,
    ExprNode * \_rarg ) [inline]
```

Subtraction expression constructor.

Parameters

_larg	Left argument of the expression.
_rarg	Right argument of the expression.

5.26.3 Member Function Documentation

5.26.3.1 eval()

```
int ExprSub::eval (
    Environment & env,
```

```
GlobalModelGenerator * generator,  
GlobalState * globalState ) [virtual]
```

Calculates the expression value.

Parameters

<i>env</i>	Environment values.
------------	---------------------

Returns

Returns an integer.

Implements [ExprNode](#).

The documentation for this class was generated from the following files:

- [expressions.hpp](#)
- [expressions.cc](#)

5.27 Formula Struct Reference

Public Attributes

- set< [Agent](#) * > **coalition**
Coalition of [Agent](#) from the formula.
- vector< [ExprNode](#) * > * **p**
- bool **isF**
- bool **isCTL**

The documentation for this struct was generated from the following file:

- [Types.hpp](#)

5.28 FormulaTemplate Struct Reference

Contains a template for coalition of [Agent](#) as string from the formula.

```
#include <Types.hpp>
```

Public Attributes

- set< string > * **coalition**
- vector< [ExprNode](#) * > * **formula**
- bool **isF**

5.28.1 Detailed Description

Contains a template for coalition of [Agent](#) as string from the formula.

The documentation for this struct was generated from the following file:

- [Types.hpp](#)

5.29 GlobalModel Struct Reference

Represents a global model, containing agents and a formula.

```
#include <GlobalModel.hpp>
```

Public Attributes

- `vector< Agent * > agents`
Pointers to all agents in a model.
- `Formula * formula`
A pointer to a [Formula](#).
- `GlobalState * initState`
Pointer to the initial state of the model.
- `vector< GlobalState * > globalStates`
Every [GlobalState](#) in the model.
- `map< Agent *, map< string, EpistemicClass * > > epistemicClasses`
Map of [Agent](#) pointers to a map of [EpistemicClass](#) for graph traversal.
- `map< Agent *, map< string, set< GlobalState * > > > epistemicClassesKnowledge`
Map of [Agent](#) pointers to a map of [EpistemicClass](#) for knowledge checks.

5.29.1 Detailed Description

Represents a global model, containing agents and a formula.

The documentation for this struct was generated from the following file:

- [GlobalModel.hpp](#)

5.30 GlobalModelGenerator Class Reference

Stores the local models, formula and a global model.

```
#include <GlobalModelGenerator.hpp>
```

Classes

- struct [GlobalStateTupleHash](#)

Public Member Functions

- [GlobalModelGenerator](#) ()
Constructor for [GlobalModelGenerator](#) class.
- [~GlobalModelGenerator](#) ()
Destructor for [GlobalModelGenerator](#) class.
- [GlobalState](#) * [initModel](#) ([LocalModels](#) *localModels, [Formula](#) *formula)
Initializes a global model from local models and a formula.
- void [expandState](#) ([GlobalState](#) *state)
Goes through all [GlobalTransition](#) in a given [GlobalState](#) and creates new [GlobalStates](#) connected to the given one.
- vector< [GlobalState](#) * > [expandStateAndReturn](#) ([GlobalState](#) *state, bool returnAnyway=false)
[GlobalModelGenerator::expandState](#) that also additionally returns a vector of newly created states.
- void [expandAllStates](#) ()
Expands the states starting from the initial [GlobalState](#) and continues until there are no more states to expand.
- void [expandAndReduceAllStates](#) ()
Expands and reduces the states starting from the initial [GlobalState](#) using a DFS-POR algorithm and continues until there are no more states to expand.
- [GlobalModel](#) * [getCurrentGlobalModel](#) ()
Get for a [GlobalModel](#) used in initialization.
- [Formula](#) * [getFormula](#) ()
Get for the [Formula](#) used in initialization.
- int [getFormulaSize](#) ()
Get size of the [Formula](#) used in initialization.
- set< [GlobalState](#) * > * [findOrCreateEpistemicClassForKnowledge](#) (vector< [LocalState](#) * > *localStates, [GlobalState](#) *globalState, [Agent](#) *agent)
Checks if a vector of [LocalState](#) is already an epistemic class for a given [Agent](#), if not, creates a new one.
- [Agent](#) * [getAgentInstanceByName](#) (string agentName)
Get a pointer to an agent by using its name.
- void [markFormulaAsIncorrect](#) ()
Sets formula to an incorrectly written state.
- bool [getFormulaCorectness](#) ()
Gets formula status.

Public Attributes

- map< [Agent](#) *, size_t > [agentIndex](#)
auxiliary variable mapping [Agent](#) pointer to its index (replace size_t with if needed later)

Protected Member Functions

- [GlobalState](#) * [generateInitState](#) ()
Generates initial state of the model from [GlobalModel](#) in memory.
- [GlobalState](#) * [generateStateFromLocalStates](#) (vector< [LocalState](#) * > *localStates, set< [LocalTransition](#) * > *viaLocalTransitions, [GlobalState](#) *prevGlobalState)
Creates a new [GlobalState](#) using some of the internally known model data and given local states, transitions that were used to get there and the previous global state.
- void [generateGlobalTransitions](#) ([GlobalState](#) *fromGlobalState, set< [LocalTransition](#) * > localTransitions, map< [Agent](#) *, vector< [LocalTransition](#) * > > transitionsByAgent)
Adds all shared global transitions to a [GlobalState](#).
- string [computeEpistemicClassHash](#) (vector< [LocalState](#) * > *localStates, [Agent](#) *agent)

- Creates a hash from a set of [LocalState](#) and an [Agent](#).
- string [computeGlobalStateHash](#) (vector< [LocalState](#) * > *localStates)
Creates a hash from a set of [LocalState](#).
- [EpistemicClass](#) * [findOrCreateEpistemicClass](#) (vector< [LocalState](#) * > *localStates, [Agent](#) *agent)
Checks if a vector of [LocalState](#) is already an epistemic class for a given [Agent](#), if not, creates a new one.
- [GlobalState](#) * [findGlobalStateInEpistemicClass](#) (vector< [LocalState](#) * > *localStates, [EpistemicClass](#) *epistemicClass)
Gets a [GlobalState](#) from an [EpistemicClass](#) if it exists in that episcemic class.

Protected Attributes

- [LocalModels](#) * [localModels](#)
[LocalModels](#) used in [initModel](#).
- [Formula](#) * [formula](#)
[Formula](#) used in [initModel](#).
- [GlobalModel](#) * [globalModel](#)
[GlobalModel](#) created in [initModel](#).
- bool [correctModel](#)
Flag holding info if model is actually correct.
- unordered_map< [GlobalState](#) *, unordered_set< int > > [candidateStateDepths](#)
Lookup map containing global states and the depths that the global state is contained in. Used for reductions.
- stack< [GlobalState](#) * > [globalModelCandidates](#)
Candidate states to be used in reductions. Used for reductions.
- stack< int > [stateDepths](#)
Saved depths of states added to [statesToExpand](#). Used for reductions.
- unordered_set< [GlobalState](#) * > [addedStates](#)
States that were added to a stack of states. Used for reductions.

5.30.1 Detailed Description

Stores the local models, formula and a global model.

5.30.2 Member Function Documentation

5.30.2.1 computeEpistemicClassHash()

```
string GlobalModelGenerator::computeEpistemicClassHash (
    vector< LocalState * > * localStates,
    Agent * agent )    [protected]
```

Creates a hash from a set of [LocalState](#) and an [Agent](#).

Parameters

localStates	Pointer to a vector of pointers of LocalState and pointer to and Agent to turn into a hash.
-----------------------------	---

Returns

Returns a string with a hash.

5.30.2.2 computeGlobalStateHash()

```
string GlobalModelGenerator::computeGlobalStateHash (
    vector< LocalState * > * localStates ) [protected]
```

Creates a hash from a set of [LocalState](#).

Parameters

<i>localStates</i>	Pointer to a vector of pointers of LocalState to turn into a hash.
--------------------	--

Returns

Returns a string with a hash.

5.30.2.3 expandAndReduceAllStates()

```
void GlobalModelGenerator::expandAndReduceAllStates ( )
```

Expands and reduces the states starting from the initial [GlobalState](#) using a DFS-POR algorithm and continues until there are no more states to expand.

Parameters

<i>depth</i>	Current depth of the recursive generation of all states.
--------------	--

5.30.2.4 expandState()

```
void GlobalModelGenerator::expandState (
    GlobalState * state )
```

Goes through all [GlobalTransition](#) in a given [GlobalState](#) and creates new GlobalStates connected to the given one.

Parameters

<i>state</i>	A state from which the expansion should start.
--------------	--

5.30.2.5 expandStateAndReturn()

```
vector< GlobalState * > GlobalModelGenerator::expandStateAndReturn (
    GlobalState * state,
    bool returnAnyway = false )
```

[GlobalModelGenerator::expandState](#) that also additionally returns a vector of newly created states.

Parameters

<i>state</i>	A state from which the expansion should start.
--------------	--

5.30.2.6 findGlobalStateInEpistemicClass()

```
GlobalState * GlobalModelGenerator::findGlobalStateInEpistemicClass (
    vector< LocalState * > * localStates,
    EpistemicClass * epistemicClass ) [protected]
```

Gets a [GlobalState](#) from an [EpistemicClass](#) if it exists in that episcemic class.

Parameters

<i>localStates</i>	Pointer to a vector of pointers to LocalState , from which will be generated a global state hash.
<i>epistemicClass</i>	Epistemic class in which to check if a GlobalState exists.

Returns

Returns a pointer to a [GlobalState](#) if it exists in given epistemic class, otherwise returns nullptr.

5.30.2.7 findOrCreateEpistemicClass()

```
EpistemicClass * GlobalModelGenerator::findOrCreateEpistemicClass (
    vector< LocalState * > * localStates,
    Agent * agent ) [protected]
```

Checks if a vector of [LocalState](#) is already an epistemic class for a given [Agent](#), if not, creates a new one.

Parameters

<i>localStates</i>	Local states from agent.
<i>agent</i>	Agent for which to check the existence of an epistemic class.

Returns

A pointer to a new or existing [EpistemicClass](#).

5.30.2.8 findOrCreateEpistemicClassForKnowledge()

```
set< GlobalState * > * GlobalModelGenerator::findOrCreateEpistemicClassForKnowledge (
    vector< LocalState * > * localStates,
    GlobalState * global,
    Agent * agent )
```

Checks if a vector of [LocalState](#) is already an epistemic class for a given [Agent](#), if not, creates a new one.

Parameters

<i>localStates</i>	Local states from agent.
<i>agent</i>	Agent for which to check the existence of an epistemic class.

Returns

A pointer to a new or existing [EpistemicClass](#).

5.30.2.9 generateGlobalTransitions()

```
void GlobalModelGenerator::generateGlobalTransitions (
    GlobalState * fromGlobalState,
    set< LocalTransition * > localTransitions,
    map< Agent *, vector< LocalTransition * >> transitionsByAgent ) [protected]
```

Adds all shared global transitions to a [GlobalState](#).

Parameters

<i>fromGlobalState</i>	Global state to add transitions to.
<i>localTransitions</i>	Initially empty, available local transitions by each agent from transitionsByAgent.
<i>transitionsByAgent</i>	Mapped transitions to an agent, only with transitions available for the agent at this moment.

5.30.2.10 generateInitState()

```
GlobalState * GlobalModelGenerator::generateInitState ( ) [protected]
```

Generates initial state of the model from [GlobalModel](#) in memory.

Returns

Returns a pointer to an initial [GlobalState](#).

5.30.2.11 generateStateFromLocalStates()

```
GlobalState * GlobalModelGenerator::generateStateFromLocalStates (
    vector< LocalState * > * localStates,
    set< LocalTransition * > * viaLocalTransitions,
    GlobalState * prevGlobalState ) [protected]
```

Creates a new [GlobalState](#) using some of the internally known model data and given local states, transitions that were used to get there and the previous global state.

Parameters

<i>localStates</i>	LocalStates from which the new GlobalState will be built.
<i>viaLocalTransitions</i>	Pointer to a set of pointers to LocalTransition from which the changes in variables, as a result of traversing through the transition, will be made in a new GlobalState .
<i>prevGlobalState</i>	Pointer to GlobalState from which all persistent variables will be copied over from to the new GlobalState .

Returns

Returns a pointer to a new or already existing in the same epistemic class [GlobalModel](#).

5.30.2.12 getAgentInstanceByName()

```
Agent * GlobalModelGenerator::getAgentInstanceByName (
    string agentName )
```

Get a pointer to an agent by using its name.

Parameters

<i>agentName</i>	Name of an Agent that we want to get its instance.
------------------	--

Returns

Pointer to an [Agent](#).

5.30.2.13 getCurrentGlobalModel()

```
GlobalModel * GlobalModelGenerator::getCurrentGlobalModel ( )
```

Get for a [GlobalModel](#) used in initialization.

Returns

Returns a pointer to a global model.

5.30.2.14 getFormula()

```
Formula * GlobalModelGenerator::getFormula ( )
```

Get for the [Formula](#) used in initialization.

Returns

Returns a pointer to the formula structure.

5.30.2.15 getFormulaCorectness()

```
bool GlobalModelGenerator::getFormulaCorectness ( )
```

Gets formula status.

Returns

True if formula is written correctly, false otherwise.

5.30.2.16 getFormulaSize()

```
int GlobalModelGenerator::getFormulaSize ( )
```

Get size of the [Formula](#) used in initialization.

Returns

Returns the formula size.

5.30.2.17 initModel()

```
GlobalState * GlobalModelGenerator::initModel (
    LocalModels * localModels,
    Formula * formula )
```

Initializes a global model from local models and a formula.

Parameters

<i>localModels</i>	Pointer to LocalModels that will construct a global model.
<i>formula</i>	Pointer to a Formula to include into the model.

Returns

Returns a pointer to initial state of the global model.

The documentation for this class was generated from the following files:

- [GlobalModelGenerator.hpp](#)
- [GlobalModelGenerator.cpp](#)

5.31 GlobalState Struct Reference

Represents a single global state.

```
#include <GlobalState.hpp>
```

Public Member Functions

- `std::string toString (string indent="")`
Debug information on the given [GlobalState](#).
- `map< string, int > getGlobalStateEnvironment ()`
Get for the environment of a given global state.

Public Attributes

- `string hash`
Hash of the global state used in quick checks if the states are in the same epistemic class.
- `map< Agent *, EpistemicClass * > epistemicClasses`
Map of agents and the epistemic classes that belongs to the respective agent.
- `bool isExpanded`
If false, the state can be still expanded, potentially creating new states, otherwise the expansion of the state already occurred and is not necessary.
- `GlobalStateVerificationStatus verificationStatus`
Current verification status of this state.
- `set< GlobalTransition * > globalTransitions`
Every [GlobalTransition](#) in the model.
- `vector< LocalState * > localStatesProjection`
Local states of each agent that define this global state.
- `map< Agent *, set< GlobalState * > * > epistemicClassesAllAgents`
States in the same epistemic class as the current one, for KBC.

5.31.1 Detailed Description

Represents a single global state.

5.31.2 Member Function Documentation

5.31.2.1 `getGlobalStateEnvironment()`

```
map< string, int > GlobalState::getGlobalStateEnvironment ( )
```

Get for the environment of a given global state.

Returns

A map of variable names and their values for the current global state.

5.31.2.2 `toString()`

```
std::string GlobalState::toString (
    string indent = "" )
```

Debug information on the given [GlobalState](#).

Parameters

<i>indent</i>	- optional indentation string
---------------	-------------------------------

Returns

[GlobalState](#) data

[GlobalState](#) data

The documentation for this struct was generated from the following files:

- [GlobalState.hpp](#)
- [GlobalState.cpp](#)

5.32 GlobalModelGenerator::GlobalStateTupleHash Struct Reference

Public Member Functions

- `size_t operator()` (const tuple< [GlobalState](#) *, int > &t) const

The documentation for this struct was generated from the following file:

- [GlobalModelGenerator.hpp](#)

5.33 GlobalTransition Struct Reference

Represents a single global transition.

```
#include <GlobalTransition.hpp>
```

Public Member Functions

- string **joinLocalTransitionNames** (char sep=';')

Public Attributes

- uint32_t **id**
Identifier of the transition.
- bool **isInvalidDecision**
Marks if the transition is invalid, true if there is no point in traversing that transition, otherwise false.
- [GlobalState](#) * **from**
Binding to a [GlobalState](#) from which this transition goes from.
- [GlobalState](#) * **to**
Binding to a [GlobalState](#) from which this transition goes to.
- set< [LocalTransition](#) * > **localTransitions**
Local transitions that define this global transition. A single transition or more in case of shared transitions.

Static Public Attributes

- static atomic_uint32_t **next_id**

5.33.1 Detailed Description

Represents a single global transition.

The documentation for this struct was generated from the following files:

- [GlobalTransition.hpp](#)
- [GlobalTransition.cpp](#)

5.34 HistoryDbg Class Reference

Stores history and allows displaying it to the console.

```
#include <Verification.hpp>
```

Public Member Functions

- [HistoryDbg](#) ()
A constructor for [HistoryDbg](#).
- [~HistoryDbg](#) ()
A destructor for [HistoryDbg](#).
- void [addEntry](#) ([HistoryEntry](#) *entry)
Adds a [HistoryEntry](#) to the debug history.
- void [markEntry](#) ([HistoryEntry](#) *entry, char chr)
Marks an entry in the debug history with a char.
- void [print](#) (string prefix)
Prints every entry from the algorithm's path.
- [HistoryEntry](#) * [cloneEntry](#) ([HistoryEntry](#) *entry)
Checks if the [HistoryEntry](#) pointer exists in the debug history.

Public Attributes

- vector< pair< [HistoryEntry](#) *, char > > [entries](#)
A pair of history entries and a char marking history type.

5.34.1 Detailed Description

Stores history and allows displaying it to the console.

5.34.2 Member Function Documentation

5.34.2.1 addEntry()

```
void HistoryDbg::addEntry (
    HistoryEntry * entry )
```

Adds a [HistoryEntry](#) to the debug history.

Parameters

<i>entry</i>	A pointer to the HistoryEntry that will be added to the history.
--------------	--

5.34.2.2 cloneEntry()

```
HistoryEntry * HistoryDbg::cloneEntry (
    HistoryEntry * entry )
```

Checks if the [HistoryEntry](#) pointer exists in the debug history.

Parameters

<i>entry</i>	A pointer to a HistoryEntry to be checked.
--------------	--

Returns

Identity function if the entry is in history, otherwise returns nullptr.

5.34.2.3 markEntry()

```
void HistoryDbg::markEntry (
    HistoryEntry * entry,
    char chr )
```

Marks an entry in the debug history with a char.

Parameters

<i>entry</i>	A pointer to a HistoryEntry that is supposed to be marked.
<i>chr</i>	A char that will be made into a pair with a HistoryEntry .

5.34.2.4 print()

```
void HistoryDbg::print (
    string prefix )
```

Prints every entry from the algorithm's path.

Parameters

<i>prefix</i>	A prefix string to append to the front of every entry.
---------------	--

The documentation for this class was generated from the following files:

- [Verification.hpp](#)
- [Verification.cpp](#)

5.35 HistoryEntry Struct Reference

Structure used to save model traversal history.

```
#include <Verification.hpp>
```

Public Member Functions

- string [toString](#) ()
Converts [HistoryEntry](#) to string.

Public Attributes

- [HistoryEntryType](#) type
Type of the history record.
- [GlobalState](#) * [globalState](#)
Saved global state.
- [GlobalTransition](#) * [decision](#)
Selected transition.
- bool [globalTransitionControlled](#)
Is the transition controlled by an agent in coalition.
- GlobalStateVerificationStatus [prevStatus](#)
Previous model verification state.
- GlobalStateVerificationStatus [newStatus](#)
Next model verification state.
- int [depth](#)
Recursion depth.
- [HistoryEntry](#) * [prev](#)
Pointer to the previous [HistoryEntry](#).
- [HistoryEntry](#) * [next](#)
Pointer to the next [HistoryEntry](#).

5.35.1 Detailed Description

Structure used to save model traversal history.

5.35.2 Member Function Documentation

5.35.2.1 [toString\(\)](#)

```
string HistoryEntry::toString ( ) [inline]
```

Converts [HistoryEntry](#) to string.

Returns

A string with the description of this history record.

The documentation for this struct was generated from the following file:

- [Verification.hpp](#)

5.36 LocalModels Struct Reference

Represents a single local model, contains all agents and variables.

```
#include <Types.hpp>
```

Public Attributes

- `vector< Agent * > agents`
A vector of agents for the current model.

5.36.1 Detailed Description

Represents a single local model, contains all agents and variables.

The documentation for this struct was generated from the following file:

- [Types.hpp](#)

5.37 LocalState Class Reference

Represents a single [LocalState](#), containing id, name and internal variables.

```
#include <LocalState.hpp>
```

Public Member Functions

- `bool compare (LocalState *state)`
Function comparing two states.
- `string toString (string indent="")`
Debug information on the given [LocalState](#).

Public Attributes

- `uint32_t id`
State identifier.
- `string name`
State name.
- `map< string, int > environment`
Local variables as a name and their current values.
- `Agent * agent`
Binding to an [Agent](#).
- `set< LocalTransition * > localTransitions`
Binding to the set of [LocalTransition](#).
- `set< GlobalState * > epistemicGlobalStates`
Binding to the set of [GlobalState](#) that this [LocalState](#) belongs to.

5.37.1 Detailed Description

Represents a single [LocalState](#), containing id, name and internal variables.

5.37.2 Member Function Documentation

5.37.2.1 `compare()`

```
bool LocalState::compare (
    LocalState * state )
```

Function comparing two states.

Parameters

<i>state</i>	A pointer to LocalState to which this state should be compared to.
--------------	--

Returns

Returns true if the current [LocalState](#) is the same as the passed one, otherwise false.

5.37.2.2 `toString()`

```
string LocalState::toString (
    string indent = "" )
```

Debug information on the given [LocalState](#).

Parameters

<i>indent</i>	- optional indentation string
---------------	-------------------------------

Returns

[LocalState](#) data

[LocalState](#) data

The documentation for this class was generated from the following files:

- [LocalState.hpp](#)
- [LocalState.cpp](#)

5.38 LocalStateTemplate Class Reference

A template for the local state.

```
#include <nodes.hpp>
```

Public Attributes

- string [name](#)
Name of the local state.
- set< [TransitionTemplate](#) * > [transitions](#)
Local transitions going out from this state.

5.38.1 Detailed Description

A template for the local state.

The documentation for this class was generated from the following file:

- [nodes.hpp](#)

5.39 LocalTransition Struct Reference

Represents a single local transition, containing id, global name, local name, is shared and count of the appearances.

```
#include <LocalTransition.hpp>
```

Public Attributes

- int [id](#)
Identifier of the transition.
- string [name](#)
Name of the transition (global).
- string [localName](#)
Name of the transition (local).
- bool [isShared](#)
Is the transition appearing somewhere else, true if yes, false if no.
- int [sharedCount](#)
Count of recurring appearances of this transition.
- set< [Condition](#) * > [conditions](#)
Conditions that have to be fulfilled for the transition to be available.
- [Agent](#) * [agent](#)
Binding to an [Agent](#).
- [LocalState](#) * [from](#)
Binding to a [LocalState](#) from which this transition goes from.
- [LocalState](#) * [to](#)
Binding to a [LocalState](#) from which this transition goes to.

5.39.1 Detailed Description

Represents a single local transition, containing id, global name, local name, is shared and count of the appearances.

The documentation for this struct was generated from the following file:

- LocalTransition.hpp

5.40 ModelParser Class Reference

A parser for converting a text file into a model.

```
#include <ModelParser.hpp>
```

Public Member Functions

- [ModelParser](#) ()
ModelParser constructor.
- [~ModelParser](#) ()
ModelParser destructor.
- tuple< [LocalModels](#), [Formula](#) > [parse](#) (string fileName)
Parses a file with given name into a usable model.
- tuple< [LocalModels](#), [Formula](#) > [parseAndOverwriteFormula](#) (string fileName, string s)

5.40.1 Detailed Description

A parser for converting a text file into a model.

5.40.2 Member Function Documentation

5.40.2.1 parse()

```
tuple< LocalModels, Formula > ModelParser::parse (
    string fileName )
```

Parses a file with given name into a usable model.

Parameters

<i>fileName</i>	Name of the file to be converted into a model.
-----------------	--

Returns

Pointer to a model created from a given file.

The documentation for this class was generated from the following files:

- [ModelParser.hpp](#)
- [ModelParser.cc](#)

5.41 TestVerif Class Reference

Public Member Functions

- **TestVerif** (string path)
- **TestVerif** (string path, bool ok)
- bool **verify** (string path, [GlobalModelGenerator](#) *generator)
- bool **verifyFull** (string path, [GlobalModelGenerator](#) *generator)
- bool **reducedVerify** (string path, [GlobalModelGenerator](#) *generator)

Public Attributes

- [GlobalModelGenerator](#) * **generator** = new [GlobalModelGenerator](#)()
- bool **result**
- string **knowledge**
- string **hartley**
- int **hCoeff**
- bool **le**

The documentation for this class was generated from the following file:

- [config.h](#)

5.42 TransitionTemplate Class Reference

Represents a meta-transition.

```
#include <nodes.hpp>
```

Public Member Functions

- [TransitionTemplate](#) (int _shared, string _patternName, string _matchName, string _startState, string _endState, [ExprNode](#) *_cond, set< [Assignment](#) * > *_assign)
[TransitionTemplate](#) constructor.

Public Attributes

- int [shared](#)
Needed amount of needed agents. -1 if not shared.
- string [patternName](#)
Name of the pattern.
- string [matchName](#)
Global name for shared transitions.
- string [startState](#)
Start state name.
- string [endState](#)
End state name.
- [ExprNode](#) * [condition](#)
[Condition](#) expression that has do be fulfilled in that transition.
- set< [Assignment](#) * > * [assignments](#)
Set of assignments.

5.42.1 Detailed Description

Represents a meta-transition.

5.42.2 Constructor & Destructor Documentation

5.42.2.1 TransitionTemplate()

```
TransitionTemplate::TransitionTemplate (
    int _shared,
    string _patternName,
    string _matchName,
    string _startState,
    string _endState,
    ExprNode * _cond,
    set< Assignment * > * _assign ) [inline]
```

[TransitionTemplate](#) constructor.

Parameters

_shared	Needed amount of needed agents. -1 if not shared.
_patternName	Name of the pattern.
_matchName	Global name for shared transitions.
_startState	Start state name.
_endState	End state name.
_cond	Condition expression that has do be fulfilled in that transition.
_assign	Set of assignments.

The documentation for this class was generated from the following file:

- [nodes.hpp](#)

5.43 Var Struct Reference

Represents a variable in the model, containing name, initial value and persistence.

```
#include <Types.hpp>
```

Public Attributes

- string [name](#)
Variable name.
- int [initialValue](#)
Initial value of the variable.
- bool [persistent](#)
True if variable is persistent, i.e. it should appear in all states in the model, false otherwise.
- [Agent](#) * [agent](#)
Reference to an agent, to which this variable belongs to.

5.43.1 Detailed Description

Represents a variable in the model, containing name, initial value and persistence.

The documentation for this struct was generated from the following file:

- [Types.hpp](#)

5.44 Verification Class Reference

A class that verifies if the model fulfills the formula. Also can do some operations on decision history.

```
#include <Verification.hpp>
```

Public Member Functions

- [Verification](#) ([GlobalModelGenerator](#) *[generator](#))
Constructor for [Verification](#).
- [~Verification](#) ()
Destructor for [Verification](#).
- bool [verify](#) ()
Starts the process of formula verification on a model.
- void [historyDecisionsERR](#) ()
Prints out the first ERR path.

Protected Member Functions

- bool `verifyLocalStates` (vector< `LocalState` * > *localStates, `GlobalState` *globalState)
Verifies a set of `LocalState` that a `GlobalState` is composed of with a hardcoded formula.
- bool `verifyGlobalState` (`GlobalState` *globalState, int depth)
Recursively verifies `GlobalState`.
- bool `isGlobalTransitionControlledByCoalition` (`GlobalTransition` *globalTransition)
Checks if any of the `LocalTransition` in a given `GlobalTransition` has an `Agent` in a coalition in the formula.
- bool `isAgentInCoalition` (`Agent` *agent)
Checks if the `Agent` is in a coalition based on the formula in a `GlobalModelGenerator`.
- `EpistemicClass` * `getEpistemicClassForGlobalState` (`GlobalState` *globalState)
Gets the `EpistemicClass` for the agent in passed `GlobalState`, i.e. transitions from indistinguishable state from certain other states for an agent to other states.
- bool `areGlobalStatesInTheSameEpistemicClass` (`GlobalState` *globalState1, `GlobalState` *globalState2)
Compares two `GlobalState` and checks if their `EpistemicClass` is the same.
- void `addHistoryDecision` (`GlobalState` *globalState, `GlobalTransition` *ecision)
Creates a `HistoryEntry` of the type `DECISION` and puts it on top of the stack of the decision history.
- void `addHistoryStateStatus` (`GlobalState` *globalState, `GlobalStateVerificationStatus` prevStatus, `GlobalStateVerificationStatus` newStatus)
Creates a `HistoryEntry` of the type `STATE_STATUS` and puts it to the top of the decision history.
- void `addHistoryContext` (`GlobalState` *globalState, int depth, `GlobalTransition` *decision, bool globalTransitionControlled)
Creates a `HistoryEntry` of the type `CONTEXT` and puts it to the top of the decision history.
- void `addHistoryMarkDecisionAsInvalid` (`GlobalState` *globalState, `GlobalTransition` *decision)
Creates a `HistoryEntry` of the type `MARK_DECISION_AS_INVALID` and puts it to the top of the decision history.
- `HistoryEntry` * `newHistoryMarkDecisionAsInvalid` (`GlobalState` *globalState, `GlobalTransition` *decision)
Creates a `HistoryEntry` of the type `MARK_DECISION_AS_INVALID` and returns it.
- bool `revertLastDecision` (int depth)
Reverts `GlobalState` and history to the previous decision state.
- void `undoLastHistoryEntry` (bool freeMemory)
Removes the top entry of the history stack.
- void `undoHistoryUntil` (`HistoryEntry` *historyEntry, bool inclusive, int depth)
Rolls back the history entries up to the certain `HistoryEntry`.
- void `printCurrentHistory` (int depth)
Prints current history to the console.
- bool `equivalentGlobalTransitions` (`GlobalTransition` *globalTransition1, `GlobalTransition` *globalTransition2)
Checks if two global transitions are made up of the same local transitions.
- bool `checkUncontrolledSet` (set< `GlobalTransition` * > uncontrolledGlobalTransitions, `GlobalState` *globalState, int depth, bool hasOmittedTransitions)
Verifies if each transition from a given state yields a correct result.
- bool `verifyTransitionSets` (set< `GlobalTransition` * > controlledGlobalTransitions, set< `GlobalTransition` * > uncontrolledGlobalTransitions, `GlobalState` *globalState, int depth, bool hasOmittedTransitions, bool isFMode)
Checks if given transition sets are able to fulfill the formula for its given epistemic class.
- bool `restoreHistory` (`GlobalState` *globalState, `GlobalTransition` *globalTransition, int depth, bool controlled)
Restores the decisions made for a given global state and transition in current recursion depth.

Protected Attributes

- [TraversalMode](#) `mode`
Current mode of model traversal.
- [GlobalState](#) * [revertToGlobalState](#)
Global state to which revert will rollback to.
- `stack` < [HistoryEntry](#) * > [historyToRestore](#)
A history of decisions to be rolled back.
- [GlobalModelGenerator](#) * [generator](#)
Holds current model and formula.
- [HistoryEntry](#) * [historyStart](#)
Pointer to the start of model traversal history.
- [HistoryEntry](#) * [historyEnd](#)
Pointer to the end of model traversal history.

5.44.1 Detailed Description

A class that verifies if the model fulfills the formula. Also can do some operations on decision history.

5.44.2 Constructor & Destructor Documentation

5.44.2.1 Verification()

```
Verification::Verification (
    GlobalModelGenerator * generator )
```

Constructor for [Verification](#).

Parameters

<code>generator</code>	Pointer to GlobalModelGenerator
------------------------	---

5.44.3 Member Function Documentation

5.44.3.1 addHistoryContext()

```
void Verification::addHistoryContext (
    GlobalState * globalState,
    int depth,
    GlobalTransition * decision,
    bool globalTransitionControlled ) [protected]
```

Creates a [HistoryEntry](#) of the type CONTEXT and puts it to the top of the decision history.

Parameters

<i>globalState</i>	Pointer to a GlobalState of the model.
<i>depth</i>	Depth of the recursion of the validation algorithm.
<i>decision</i>	Pointer to a transition GlobalTransition selected by the algorithm.
<i>globalTransitionControlled</i>	True if the GlobalTransition is in the set of global transitions controlled by a coalition and it is not a fixed global transition.

5.44.3.2 addHistoryDecision()

```
void Verification::addHistoryDecision (
    GlobalState * globalState,
    GlobalTransition * decision ) [protected]
```

Creates a [HistoryEntry](#) of the type DECISION and puts it on top of the stack of the decision history.

Parameters

<i>globalState</i>	Pointer to a GlobalState of the model.
<i>decision</i>	Pointer to a GlobalTransition that is to be recorded in the decision history.

5.44.3.3 addHistoryMarkDecisionAsInvalid()

```
void Verification::addHistoryMarkDecisionAsInvalid (
    GlobalState * globalState,
    GlobalTransition * decision ) [protected]
```

Creates a [HistoryEntry](#) of the type MARK_DECISION_AS_INVALID and puts it to the top of the decision history.

Parameters

<i>globalState</i>	Pointer to a GlobalState of the model.
<i>decision</i>	Pointer to a transition GlobalTransition selected by the algorithm.

5.44.3.4 addHistoryStateStatus()

```
void Verification::addHistoryStateStatus (
    GlobalState * globalState,
    GlobalStateVerificationStatus prevStatus,
    GlobalStateVerificationStatus newStatus ) [protected]
```

Creates a [HistoryEntry](#) of the type STATE_STATUS and puts it to the top of the decision history.

Parameters

<i>globalState</i>	Pointer to a GlobalState of the model.
<i>prevStatus</i>	Previous GlobalStateVerificationStatus to be logged.
<i>newStatus</i>	New GlobalStateVerificationStatus to be logged.

5.44.3.5 areGlobalStatesInTheSameEpistemicClass()

```
bool Verification::areGlobalStatesInTheSameEpistemicClass (
    GlobalState * globalState1,
    GlobalState * globalState2 ) [protected]
```

Compares two [GlobalState](#) and checks if their [EpistemicClass](#) is the same.

Parameters

<i>globalState1</i>	Pointer to the first GlobalState .
<i>globalState2</i>	Pointer to the second GlobalState .

Returns

Returns true if the [EpistemicClass](#) is the same for both of the [GlobalState](#). Returns false if they are different or at least one of them has no [EpistemicClass](#).

5.44.3.6 checkUncontrolledSet()

```
bool Verification::checkUncontrolledSet (
    set< GlobalTransition * > uncontrolledGlobalTransitions,
    GlobalState * globalState,
    int depth,
    bool hasOmittedTransitions ) [protected]
```

Verifies if each transition from a given state yields a correct result.

Parameters

<i>uncontrolledGlobalTransitions</i>	A set of global transitions to be checked.
<i>globalState</i>	Currently processed global state.
<i>depth</i>	Current recursion depth.
<i>hasOmittedTransitions</i>	Flag with the information about skipped unneeded transitions.

Returns

Returns true if every transition yields a correct result, false otherwise.

5.44.3.7 equivalentGlobalTransitions()

```
bool Verification::equivalentGlobalTransitions (
    GlobalTransition * globalTransition1,
    GlobalTransition * globalTransition2 ) [protected]
```

Checks if two global transitions are made up of the same local transitions.

Parameters

<i>globalTransition1</i>	First global transition to compare.
<i>globalTransition2</i>	Second global transition to compare.

Returns

True if the two global transitions have the same local transitions, false otherwise.

5.44.3.8 getEpistemicClassForGlobalState()

```
EpistemicClass * Verification::getEpistemicClassForGlobalState (
    GlobalState * globalState ) [protected]
```

Gets the [EpistemicClass](#) for the agent in passed [GlobalState](#), i.e. transitions from indistinguishable state from certain other states for an agent to other states.

Parameters

<i>globalState</i>	Pointer to a GlobalState of the model.
--------------------	--

Returns

Pointer to the [EpistemicClass](#) that a coalition of agents from the formula belong to. If there is no such [EpistemicClass](#), returns false.

5.44.3.9 isAgentInCoalition()

```
bool Verification::isAgentInCoalition (
    Agent * agent ) [protected]
```

Checks if the [Agent](#) is in a coalition based on the formula in a [GlobalModelGenerator](#).

Parameters

<i>agent</i>	Pointer to an Agent that is to be checked.
--------------	--

Returns

Returns true if the [Agent](#) is in a coalition, otherwise returns false.

5.44.3.10 isGlobalTransitionControlledByCoalition()

```
bool Verification::isGlobalTransitionControlledByCoalition (  
    GlobalTransition * globalTransition ) [protected]
```

Checks if any of the [LocalTransition](#) in a given [GlobalTransition](#) has an [Agent](#) in a coalition in the formula.

Parameters

<i>globalTransition</i>	Pointer to a GlobalTransition in a model.
-------------------------	---

Returns

Returns true if the [Agent](#) is in coalition in the formula, otherwise returns false.

5.44.3.11 newHistoryMarkDecisionAsInvalid()

```
HistoryEntry * Verification::newHistoryMarkDecisionAsInvalid (  
    GlobalState * globalState,  
    GlobalTransition * decision ) [protected]
```

Creates a [HistoryEntry](#) of the type MARK_DECISION_AS_INVALID and returns it.

Parameters

<i>globalState</i>	Pointer to a GlobalState of the model.
<i>decision</i>	Pointer to a transition GlobalTransition selected by the algorithm.

Returns

Returns pointer to a new [HistoryEntry](#).

5.44.3.12 printCurrentHistory()

```
void Verification::printCurrentHistory (
    int depth ) [protected]
```

Prints current history to the console.

Parameters

<i>depth</i>	Integer that will be multiplied by 4 and appended as a prefix to the optional debug log.
--------------	--

5.44.3.13 restoreHistory()

```
bool Verification::restoreHistory (
    GlobalState * globalState,
    GlobalTransition * globalTransition,
    int depth,
    bool controlled ) [protected]
```

Restores the decisions made for a given global state and transition in current recursion depth.

Parameters

<i>globalState</i>	Currently processed global state.
<i>globalTransition</i>	Previously selected global transition from the given state to mark as invalid.
<i>depth</i>	Current recursion depth.
<i>controlled</i>	Flag with the information about current type of transition. True if controlled, false if uncontrolled.

Returns

Returns true if current top of the history entires matches with

5.44.3.14 revertLastDecision()

```
bool Verification::revertLastDecision (
    int depth ) [protected]
```

Reverts [GlobalState](#) and history to the previous decision state.

Parameters

<i>depth</i>	Integer that will be multiplied by 4 and appended as a prefix to the optional debug log.
--------------	--

Returns

Returns true if rollback is successful, otherwise returns false.

5.44.3.15 undoHistoryUntil()

```
void Verification::undoHistoryUntil (
    HistoryEntry * historyEntry,
    bool inclusive,
    int depth ) [protected]
```

Rolls back the history entries up to the certain [HistoryEntry](#).

Parameters

<i>historyEntry</i>	Pointer to a HistoryEntry that the history has to be rolled back to.
<i>inclusive</i>	True if the rollback has to remove the specified entry too.
<i>depth</i>	Integer that will be multiplied by 4 and appended as a prefix to the optional debug log.

5.44.3.16 undoLastHistoryEntry()

```
void Verification::undoLastHistoryEntry (
    bool freeMemory ) [protected]
```

Removes the top entry of the history stack.

Parameters

<i>freeMemory</i>	True if the entry has to be removed from memory.
-------------------	--

5.44.3.17 verify()

```
bool Verification::verify ( )
```

Starts the process of formula verification on a model.

Returns

Returns true if the verification is PENDING or VERIFIED_OK, otherwise returns false.

5.44.3.18 verifyGlobalState()

```
bool Verification::verifyGlobalState (
    GlobalState * globalState,
    int depth ) [protected]
```

Recursively verifies [GlobalState](#).

Parameters

<i>globalState</i>	Pointer to a GlobalState of the model.
<i>depth</i>	Current depth of the recursion.

Returns

Returns true if the verification is PENDING or VERIFIED_OK, otherwise returns false.

5.44.3.19 verifyLocalStates()

```
bool Verification::verifyLocalStates (
    vector< LocalState * > * localStates,
    GlobalState * globalState ) [protected]
```

Verifies a set of [LocalState](#) that a [GlobalState](#) is composed of with a hardcoded formula.

Parameters

<i>localStates</i>	A pointer to a set of pointers to LocalState .
--------------------	--

Returns

Returns true if there is a [LocalState](#) with a specific set of values, fulfilling the criteria, otherwise returns false.

5.44.3.20 verifyTransitionSets()

```
bool Verification::verifyTransitionSets (
    set< GlobalTransition * > controlledGlobalTransitions,
    set< GlobalTransition * > uncontrolledGlobalTransitions,
    GlobalState * globalState,
    int depth,
    bool hasOmittedTransitions,
    bool isFMode ) [protected]
```

Checks if given transition sets are able to fulfill the formula for its given epistemic class.

Parameters

<i>controlledGlobalTransitions</i>	Set of controlled transitions in the current global state.
<i>uncontrolledGlobalTransitions</i>	Set of uncontrolled transitions in the current global state.
<i>globalState</i>	Currently processed global state.
<i>depth</i>	Current recursion depth.
<i>hasOmittedTransitions</i>	Flag with the information about skipped unneeded transitions.

Returns

True if there is a correct choice for an agent to take, false otherwise.

The documentation for this class was generated from the following files:

- [Verification.hpp](#)
- [Verification.cpp](#)

5.45 yy_buffer_state Struct Reference

Public Attributes

- FILE * **yy_input_file**
- char * **yy_ch_buf**
- char * **yy_buf_pos**
- int **yy_buf_size**
- int **yy_n_chars**
- int **yy_is_our_buffer**
- int **yy_is_interactive**
- int **yy_at_bol**
- int [yy_bs_lineno](#)
- int [yy_bs_column](#)
- int **yy_fill_buffer**
- int **yy_buffer_status**

5.45.1 Member Data Documentation

5.45.1.1 yy_bs_column

```
int yy_buffer_state::yy_bs_column
```

The column count.

5.45.1.2 yy_bs_lineno

```
int yy_buffer_state::yy_bs_lineno
```

The line count.

The documentation for this struct was generated from the following file:

- scanner.c

5.46 yy_trans_info Struct Reference

Public Attributes

- flex_int32_t **yy_verify**
- flex_int32_t **yy_nxt**

The documentation for this struct was generated from the following file:

- scanner.c

5.47 yyallocc Union Reference

Public Attributes

- yy_state_t **yyss_alloc**
- YYSTYPE **yyvs_alloc**

The documentation for this union was generated from the following file:

- parser.c

5.48 YYSTYPE Union Reference

Public Attributes

- set< class [AgentTemplate](#) * > * **model**
- class [ExprNode](#) * **expr**
- class [Assignment](#) * **assign**
- class [TransitionTemplate](#) * **trans**
- class [AgentTemplate](#) * **agent**
- set< class [Assignment](#) * > * **assignSet**
- char * **ident**
- set< string > * **identSet**
- int **val**
- vector< class [ExprNode](#) * > * **condList**

The documentation for this union was generated from the following file:

- parser.h

Chapter 6

File Documentation

6.1 Agent.cpp File Reference

Class of an agent. Class of an agent.

```
#include "Agent.hpp"  
#include "LocalState.hpp"
```

6.1.1 Detailed Description

Class of an agent. Class of an agent.

6.2 Agent.hpp File Reference

Class of an agent. Class of an agent.

```
#include "Common.hpp"
```

Classes

- class [Agent](#)

Contains all data for a single [Agent](#), including id, name and all of the agents' variables.

6.2.1 Detailed Description

Class of an agent. Class of an agent.

6.3 Common.hpp File Reference

Contains all commonly used classes and structs. Contains all commonly used classes and structs.

```
#include <map>
#include <set>
#include <stack>
#include <string>
#include <utility>
#include <vector>
#include <atomic>
#include "reader/expressions.hpp"
#include "enums/GlobalStateVerificationStatus.hpp"
#include "enums/ConditionOperator.hpp"
```

Classes

- struct [Cfg](#)

6.3.1 Detailed Description

Contains all commonly used classes and structs. Contains all commonly used classes and structs.

6.4 DotGraph.cpp File Reference

Class for drawing graphs of input models. Class used for making graph files of input models created by parsing the data and adding graphviz syntax.

```
#include "DotGraph.hpp"
#include <algorithm>
```

6.4.1 Detailed Description

Class for drawing graphs of input models. Class used for making graph files of input models created by parsing the data and adding graphviz syntax.

6.5 DotGraph.hpp File Reference

Class for drawing graphs of input models. Class used for making graph files of input models created by parsing the data and adding graphviz syntax.

```
#include "Types.hpp"
#include "Utils.hpp"
#include "reader/nodes.hpp"
#include <string>
```

Classes

- class [DotGraph](#)

Enumerations

- enum [DotGraphBase](#) { [LOCAL_MODEL](#) , [GLOBAL_MODEL](#) , [AGENT_TEMPLATE](#) }

6.5.1 Detailed Description

Class for drawing graphs of input models. Class used for making graph files of input models created by parsing the data and adding graphviz syntax.

6.5.2 Enumeration Type Documentation

6.5.2.1 DotGraphBase

enum [DotGraphBase](#)

Enumerator

LOCAL_MODEL	(unfolded) local model = TS with states and transitions
GLOBAL_MODEL	(unfolded) global model = TS with states and transitions
AGENT_TEMPLATE	(folded/compact) agent graph = sets of locations and labelled edges

6.6 EpistemicClass.hpp File Reference

Struct of an epistemic class. Struct of an epistemic class.

```
#include "Common.hpp"
```

Classes

- struct [EpistemicClass](#)
Represents a single epistemic class.

6.6.1 Detailed Description

Struct of an epistemic class. Struct of an epistemic class.

6.7 expressions.cc File Reference

Eval and helper class for expressions. Eval and helper class for expressions.

```
#include "expressions.hpp"
#include "../GlobalModelGenerator.hpp"
#include <bits/stdc++.h>
```

6.7.1 Detailed Description

Eval and helper class for expressions. Eval and helper class for expressions.

6.8 expressions.hpp File Reference

Eval and helper class for expressions. Eval and helper class for expressions.

```
#include <string>
#include <map>
#include <vector>
```

Classes

- class [ExprNode](#)
Base node for expressions.
- class [ExprConst](#)
Node for a constant.
- class [ExprIdent](#)
Node for an identifier.
- class [ExprAdd](#)
Node for addition.
- class [ExprSub](#)
Node for subtraction.
- class [ExprMul](#)
Node for multiplication.
- class [ExprDiv](#)
Node for division.
- class [ExprRem](#)
Node for modulo.
- class [ExprAnd](#)
Node for AND operator.
- class [ExprOr](#)
Node for OR operator.
- class [ExprNot](#)
Node for NOT operator.
- class [ExprEq](#)
Node for "==" operator.

- class [ExprNe](#)
Node for "!=" operator.
- class [ExprLt](#)
Node for "<" operator.
- class [ExprLe](#)
Node for "<=" operator.
- class [ExprGt](#)
Node for ">" operator.
- class [ExprGe](#)
Node for ">=" operator.
- class [ExprKnow](#)
Node for a knowledge operator.
- class [ExprHart](#)
Node for a Hartley operator.

Typedefs

- typedef map< string, int > [Environment](#)
Variable names with their values.

6.8.1 Detailed Description

Eval and helper class for expressions. Eval and helper class for expressions.

6.9 GlobalModel.hpp File Reference

Struct of a global model. Struct of a global model.

```
#include "Common.hpp"
```

Classes

- struct [GlobalModel](#)
Represents a global model, containing agents and a formula.

6.9.1 Detailed Description

Struct of a global model. Struct of a global model.

6.10 GlobalModelGenerator.cpp File Reference

Generator of a global model. Class for initializing and generating a global model.

```
#include "GlobalModelGenerator.hpp"
#include "Types.hpp"
#include "Constants.hpp"
#include <algorithm>
#include <string.h>
#include <iostream>
#include <sstream>
```

Variables

- [Cfg config](#)

6.10.1 Detailed Description

Generator of a global model. Class for initializing and generating a global model.

6.11 GlobalModelGenerator.hpp File Reference

Generator of a global model. Class for initializing and generating a global model.

```
#include "Constants.hpp"
#include "GlobalState.hpp"
#include "GlobalTransition.hpp"
#include "Agent.hpp"
#include <unordered_set>
#include <unordered_map>
#include <sstream>
```

Classes

- class [GlobalModelGenerator](#)
Stores the local models, formula and a global model.
- struct [GlobalModelGenerator::GlobalStateTupleHash](#)

6.11.1 Detailed Description

Generator of a global model. Class for initializing and generating a global model.

6.12 GlobalState.cpp File Reference

Struct representing a global state. Struct representing a global state.

```
#include "GlobalState.hpp"
#include "LocalState.hpp"
```

6.12.1 Detailed Description

Struct representing a global state. Struct representing a global state.

6.13 GlobalState.hpp File Reference

Struct representing a global state. Struct representing a global state.

```
#include "Common.hpp"
```

Classes

- struct [GlobalState](#)
Represents a single global state.

6.13.1 Detailed Description

Struct representing a global state. Struct representing a global state.

6.14 GlobalTransition.cpp File Reference

Struct representing a global transition. Struct representing a global transition.

```
#include "GlobalTransition.hpp"
#include "LocalTransition.hpp"
```

6.14.1 Detailed Description

Struct representing a global transition. Struct representing a global transition.

6.15 GlobalTransition.hpp File Reference

Struct representing a global transition. Struct representing a global transition.

```
#include "Common.hpp"
```

Classes

- struct [GlobalTransition](#)
Represents a single global transition.

6.15.1 Detailed Description

Struct representing a global transition. Struct representing a global transition.

6.16 LocalState.cpp File Reference

Class representing a local state. Class representing a local state.

```
#include "LocalState.hpp"  
#include "Agent.hpp"
```

6.16.1 Detailed Description

Class representing a local state. Class representing a local state.

6.17 LocalState.hpp File Reference

Class representing a local state. Class representing a local state.

```
#include "Common.hpp"
```

Classes

- class [LocalState](#)
Represents a single [LocalState](#), containing id, name and internal variables.

6.17.1 Detailed Description

Class representing a local state. Class representing a local state.

6.18 ModelParser.cc File Reference

A model parser. A parser for converting a text file into a model.

```
#include "ModelParser.hpp"
#include "reader/nodes.hpp"
#include <stdio.h>
#include <string.h>
#include <tuple>
#include <iostream>
```

Functions

- int **yyparse** ()
- void **yyrestart** (FILE *)
- void **set_input_string** (const char *in)

Variables

- set< [AgentTemplate](#) * > * **modelDescription**
- [FormulaTemplate](#) **formulaDescription**

6.18.1 Detailed Description

A model parser. A parser for converting a text file into a model.

6.19 nodes.cc File Reference

Parser templates. Class for setting up a new objects from a parser.

```
#include "expressions.hpp"
#include "nodes.hpp"
#include <queue>
#include <fstream>
```

6.19.1 Detailed Description

Parser templates. Class for setting up a new objects from a parser.

6.20 nodes.hpp File Reference

Parser templates. Class for setting up a new objects from a parser.

```
#include <string>
#include <set>
#include <map>
#include "expressions.hpp"
#include "../Types.hpp"
```

Classes

- class [Assignment](#)
Represents an assingment.
- class [TransitionTemplate](#)
Represents a meta-transition.
- class [LocalStateTemplate](#)
A template for the local state.
- class [AgentTemplate](#)
Represents a single agent loaded from the description from a file.

6.20.1 Detailed Description

Parser templates. Class for setting up a new objects from a parser.

6.21 Types.hpp File Reference

Custom data structures. Data structures and classes containing model data.

```
#include <map>
#include <set>
#include <stack>
#include <string>
#include <utility>
#include <vector>
#include "LocalState.hpp"
#include "LocalTransition.hpp"
#include "GlobalState.hpp"
#include "GlobalTransition.hpp"
#include "GlobalModel.hpp"
#include "EpistemicClass.hpp"
#include "Agent.hpp"
#include "reader/expressions.hpp"
#include <atomic>
```

Classes

- struct [Var](#)
Represents a variable in the model, containing name, initial value and persistence.
- struct [Condition](#)
Represents a condition for [LocalTransition](#).
- struct [FormulaTemplate](#)
Contains a template for coalition of [Agent](#) as string from the formula.
- struct [Formula](#)
- struct [LocalModels](#)
Represents a single local model, contains all agents and variables.

6.21.1 Detailed Description

Custom data structures. Data structures and classes containing model data.

6.22 Utils.cpp File Reference

Utility functions. A collection of utility functions to use in the project.

```
#include "Utils.hpp"
#include <fstream>
#include "GlobalModelGenerator.hpp"
#include <map>
#include <algorithm>
#include <iostream>
#include <cstdio>
```

Functions

- string [envToString](#) (map< string, int > env)
Converts a map of string and int to a string.
- string [agentToString](#) ([Agent](#) *agt)
Converts pointer to an Agent into a string containing name of the agent, its initial state, transitions with their local and global names, shared count and conditions.
- string [localModelsToString](#) ([LocalModels](#) *lm)
Converts pointer to the [LocalModels](#) into a string containing all [Agent](#) instances from the model, initial values of the variables and names of the persistent values.
- void [outputGlobalModel](#) ([GlobalModel](#) *globalModel)
Prints the whole [GlobalModel](#) into the console. Contains global states with hashes, local states, variables inside those states, global variables, global transitions, local transitions and epistemic classes of agents.
- unsigned long [getMemCap](#) ()
- void [loadConfigFromFile](#) (string filename)
- void [loadConfigFromArgs](#) (int argc, char **argv)
- void [tarjanVisit](#) ([LocalState](#) *v, map< int, int > *dindex, map< int, int > *lowlink, stack< [LocalState](#) * > *stack, map< int, bool > *onstack, int *depth, vector< set< [LocalState](#) * > > *comp)
Utility function for SCC-computation.
- vector< set< [LocalState](#) * > > [getLocalStatesSCC](#) ([Agent](#) *agt)
a quick implementation of a Tarjan SCC algorithm (based on DFS)
- map< [LocalState](#) *, vector< [GlobalState](#) * > > [getContextModel](#) ([Formula](#) *formula, [LocalModels](#) *localModels, [Agent](#) *agt)

Variables

- [Cfg config](#)

6.22.1 Detailed Description

Utility functions. A collection of utility functions to use in the project.

6.22.2 Function Documentation

6.22.2.1 agentToString()

```
string agentToString (
    Agent * agt )
```

Converts pointer to an Agent into a string containing name of the agent, its initial state, transitions with their local and global names, shared count and conditions.

Parameters

<i>agt</i>	Pointer to an Agent to parse into a string.
------------	---

Returns

String containing all of [Agent](#) data.

6.22.2.2 envToString()

```
string envToString (
    map< string, int > env )
```

Converts a map of string and int to a string.

Parameters

<i>env</i>	Map to be converted into a string.
------------	------------------------------------

Returns

Returns string " (first_name, second_name, ..., last_name=int_value)"

6.22.2.3 getLocalStatesSCC()

```
vector<set<LocalState*> > getLocalStatesSCC (
    Agent * agt )
```

a quick implementation of a Tarjan SCC algorithm (based on DFS)

Parameters

<i>agt</i>	- an agent whose local graph will be inspected
------------	--

Returns

localStates partition in a form of the vector, where each set correponds to a SCC

6.22.2.4 localModelsToString()

```
string localModelsToString (
    LocalModels * lm )
```

Converts pointer to the [LocalModels](#) into a string containing all [Agent](#) instances from the model, initial values of the variables and names of the persistent values.

Parameters

<i>lm</i>	Pointer to the local model to parse into a string.
-----------	--

Returns

String containing all of [LocalModels](#) data.

6.22.2.5 outputGlobalModel()

```
void outputGlobalModel (
    GlobalModel * globalModel )
```

Prints the whole [GlobalModel](#) into the console. Contains global states with hashes, local states, variables inside those states, global variables, global transitions, local transitions and epistemic classes of agents.

Parameters

<i>globalModel</i>	Pointer to a GlobalModel to print into the console.
--------------------	---

6.22.2.6 tarjanVisit()

```
void tarjanVisit (
    LocalState * v,
    map< int, int > * dindex,
    map< int, int > * lowlink,
    stack< LocalState * > * stack,
    map< int, bool > * onstack,
    int * depth,
    vector< set< LocalState * >> * comp )
```

Utility function for SCC-computatation.

Parameters

<i>v</i>	- current vertex
<i>dindex</i>	- vertex.index (alt. vertex.num)
<i>lowlink</i>	- vertex.lowlink (by df. lowest index in the same scc reachable from vertex using tree edges followed by at most one back/cross edge)
<i>stack</i>	- holds candidates for SCC
<i>onstack</i>	- used as condition for back/cross-edge case
<i>depth</i>	- next available (discovery) index
<i>comp</i>	- result of SCC partitioning

6.23 Utils.hpp File Reference

```
#include "Types.hpp"
#include "Constants.hpp"
#include <map>
#include <string>
#include <unistd.h>
#include <sys/time.h>
#include <iostream>
#include <fstream>
```

Functions

- string [envToString](#) (map< string, int > env)
Converts a map of string and int to a string.
- string [agentToString](#) (Agent *agt)
Converts pointer to an Agent into a string containing name of the agent, its initial state, transitions with their local and global names, shared count and conditions.
- string [localModelsToString](#) (LocalModels *lm)
Converts pointer to the LocalModels into a string containing all Agent instances from the model, initial values of the variables and names of the persistent values.
- void [outputGlobalModel](#) (GlobalModel *globalModel)
Prints the whole GlobalModel into the console. Contains global states with hashes, local states, variables inside those states, global variables, global transitions, local transitions and epistemic classes of agents.
- unsigned long [getMemCap](#) ()

- vector< set< [LocalState](#) * > > [getLocalStatesSCC](#) ([Agent](#) *agt)
a quick implementation of a Tarjan SCC algorithm (based on DFS)
- map< [LocalState](#) *, vector< [GlobalState](#) * > > [getContextModel](#) ([Formula](#) *formula, [LocalModels](#) *localModels, [Agent](#) *agt)
- void [loadConfigFromFile](#) (string filename="config.txt")
- void [loadConfigFromArgs](#) (int argc, char **argv)

6.23.1 Function Documentation

6.23.1.1 agentToString()

```
string agentToString (
    Agent * agt )
```

Converts pointer to an Agent into a string containing name of the agent, its initial state, transitions with their local and global names, shared count and conditions.

Parameters

<i>agt</i>	Pointer to an Agent to parse into a string.
------------	---

Returns

String containing all of [Agent](#) data.

6.23.1.2 envToString()

```
string envToString (
    map< string, int > env )
```

Converts a map of string and int to a string.

Parameters

<i>env</i>	Map to be converted into a string.
------------	------------------------------------

Returns

Returns string " (first_name, second_name, ..., last_name=int_value)"

6.23.1.3 getLocalStatesSCC()

```
vector<set<LocalState*> > getLocalStatesSCC (
    Agent * agt )
```

a quick implementation of a Tarjan SCC algorithm (based on DFS)

Parameters

<i>agt</i>	- an agent whose local graph will be inspected
------------	--

Returns

localStates partition in a form of the vector, where each set correponds to a SCC

6.23.1.4 localModelsToString()

```
string localModelsToString (
    LocalModels * lm )
```

Converts pointer to the [LocalModels](#) into a string containing all [Agent](#) instances from the model, initial values of the variables and names of the persistent values.

Parameters

<i>lm</i>	Pointer to the local model to parse into a string.
-----------	--

Returns

String containing all of [LocalModels](#) data.

6.23.1.5 outputGlobalModel()

```
void outputGlobalModel (
    GlobalModel * globalModel )
```

Prints the whole [GlobalModel](#) into the console. Contains global states with hashes, local states, variables inside those states, global variables, global transitions, local transitions and epistemic classes of agents.

Parameters

<i>globalModel</i>	Pointer to a GlobalModel to print into the console.
--------------------	---

6.24 Verification.cpp File Reference

Class for verification of the formula on a model. Class for verification of the specified formula on a specified model.

```
#include "Verification.hpp"
#include <bits/stdc++.h>
```

Macros

- `#define DEPTH_PREFIX` `string(depth * 4, '')`

Functions

- `string verStatusToStr` (`GlobalStateVerificationStatus` status)
Converts global verification status into a string.
- `void dbgVerifStatus` (`string` prefix, `GlobalState` *gs, `GlobalStateVerificationStatus` st, `string` reason)
Print a debug message of a verification status to the console.
- `void dbgHistEnt` (`string` prefix, `HistoryEntry` *h)
Print a single debug message with a history entry to the console.

Variables

- `Cfg` `config`

6.24.1 Detailed Description

Class for verification of the formula on a model. Class for verification of the specified formula on a specified model.

6.24.2 Function Documentation

6.24.2.1 dbgHistEnt()

```
void dbgHistEnt (
    string prefix,
    HistoryEntry * h )
```

Print a single debug message with a history entry to the console.

Parameters

<i>prefix</i>	A prefix string to append to the front of the entry.
<i>h</i>	A pointer to the <code>HistoryEntry</code> struct which will be printed out.

6.24.2.2 dbgVerifStatus()

```
void dbgVerifStatus (
    string prefix,
    GlobalState * gs,
    GlobalStateVerificationStatus st,
    string reason )
```

Print a debug message of a verification status to the console.

Parameters

<i>prefix</i>	A prefix string to append to the front of every entry.
<i>gs</i>	Pointer to a GlobalState .
<i>st</i>	Enum with a verification status of a global state.
<i>reason</i>	String with a reason why the function was called, e.g. "entered state", "all passed".

6.24.2.3 verStatusToStr()

```
string verStatusToStr (
    GlobalStateVerificationStatus status )
```

Converts global verification status into a string.

Parameters

<i>status</i>	Enum value to be converted.
---------------	-----------------------------

Returns

[Verification](#) status converted into a string.

6.25 Verification.hpp File Reference

```
#include <stack>
#include "Types.hpp"
#include "GlobalModelGenerator.hpp"
```

Classes

- struct [HistoryEntry](#)

Structure used to save model traversal history.

- class [HistoryDbg](#)
Stores history and allows displaying it to the console.
- class [Verification](#)
A class that verifies if the model fulfills the formula. Also can do some operations on decision history.

Enumerations

- enum [HistoryEntryType](#) { [DECISION](#) , [STATE_STATUS](#) , [CONTEXT](#) , [MARK_DECISION_AS_INVALID](#) }
HistoryEntry entry type.
- enum [TraversalMode](#) { [NORMAL](#) , [REVERT](#) , [RESTORE](#) }
Current model traversal mode.

Functions

- string [verStatusToStr](#) (GlobalStateVerificationStatus status)
Converts global verification status into a string.

6.25.1 Enumeration Type Documentation

6.25.1.1 HistoryEntryType

enum [HistoryEntryType](#)

[HistoryEntry](#) entry type.

Enumerator

DECISION	Made the decision to go to a state using a transition.
STATE_STATUS	Changed verification status.
CONTEXT	Recursion has gone deeper.
MARK_DECISION_AS_INVALID	Marking a transition as invalid.

6.25.1.2 TraversalMode

enum [TraversalMode](#)

Current model traversal mode.

Enumerator

NORMAL	Normal model traversal.
REVERT	Backtracking through recursion with state rollback.
RESTORE	Backtracking through recursion.

6.25.2 Function Documentation

6.25.2.1 verStatusToStr()

```
string verStatusToStr (
    GlobalStateVerificationStatus status )
```

Converts global verification status into a string.

Parameters

<i>status</i>	Enum value to be converted.
---------------	-----------------------------

Returns

[Verification](#) status converted into a string.

Index

- addEdge
 - DotGraph, [20](#)
- addEntry
 - HistoryDbg, [64](#)
- addHistoryContext
 - Verification, [75](#)
- addHistoryDecision
 - Verification, [76](#)
- addHistoryMarkDecisionAsInvalid
 - Verification, [76](#)
- addHistoryStateStatus
 - Verification, [76](#)
- addInitial
 - AgentTemplate, [13](#)
- addLocal
 - AgentTemplate, [14](#)
- addNode
 - DotGraph, [21](#)
- addPersistent
 - AgentTemplate, [14](#)
- addTransition
 - AgentTemplate, [15](#)
- Agent, [11](#)
 - Agent, [12](#)
 - includesState, [12](#)
- Agent.cpp, [85](#)
- Agent.hpp, [85](#)
- AGENT_TEMPLATE
 - DotGraph.hpp, [87](#)
- AgentTemplate, [12](#)
 - addInitial, [13](#)
 - addLocal, [14](#)
 - addPersistent, [14](#)
 - addTransition, [15](#)
 - generateAgent, [15](#)
 - setIdnt, [16](#)
 - setInitState, [16](#)
- agentToString
 - Utils.cpp, [96](#)
 - Utils.hpp, [99](#)
- areGlobalStatesInTheSameEpistemicClass
 - Verification, [77](#)
- assign
 - Assignment, [18](#)
- Assignment, [17](#)
 - assign, [18](#)
- Assignment, [18](#)
- Cfg, [18](#)
- checkUncontrolledSet
 - Verification, [77](#)
- cloneEntry
 - HistoryDbg, [64](#)
- Common.hpp, [86](#)
- compare
 - LocalState, [68](#)
- computeEpistemicClassHash
 - GlobalModelGenerator, [55](#)
- computeGlobalStateHash
 - GlobalModelGenerator, [56](#)
- Condition, [19](#)
- CONTEXT
 - Verification.hpp, [103](#)
- dbgHistEnt
 - Verification.cpp, [101](#)
- dbgVerifStatus
 - Verification.cpp, [102](#)
- DECISION
 - Verification.hpp, [103](#)
- DotGraph, [20](#)
 - addEdge, [20](#)
 - addNode, [21](#)
 - saveToFile, [21](#)
 - styleString, [21](#)
- DotGraph.cpp, [86](#)
- DotGraph.hpp, [86](#)
 - AGENT_TEMPLATE, [87](#)
 - DotGraphBase, [87](#)
 - GLOBAL_MODEL, [87](#)
 - LOCAL_MODEL, [87](#)
- DotGraphBase
 - DotGraph.hpp, [87](#)
- envToString
 - Utils.cpp, [96](#)
 - Utils.hpp, [99](#)
- EpistemicClass, [22](#)
- EpistemicClass.hpp, [87](#)
- equivalentGlobalTransitions
 - Verification, [78](#)
- eval
 - ExprAdd, [23](#)
 - ExprAnd, [25](#)
 - ExprConst, [26](#)
 - ExprDiv, [27](#)
 - ExprEq, [29](#)
 - ExprGe, [30](#)
 - ExprGt, [31](#)
 - ExprHart, [34](#)

- ExprIdent, 35
- ExprKnow, 38
- ExprLe, 39
- ExprLt, 40
- ExprMul, 43
- ExprNe, 44
- ExprNode, 45
- ExprNot, 46
- ExprOr, 48
- ExprRem, 49
- ExprSub, 50
- expandAndReduceAllStates
 - GlobalModelGenerator, 56
- expandState
 - GlobalModelGenerator, 56
- expandStateAndReturn
 - GlobalModelGenerator, 56
- ExprAdd, 22
 - eval, 23
 - ExprAdd, 23
- ExprAnd, 24
 - eval, 25
 - ExprAnd, 24
- ExprConst, 25
 - eval, 26
 - ExprConst, 26
- ExprDiv, 26
 - eval, 27
 - ExprDiv, 27
- ExprEq, 28
 - eval, 29
 - ExprEq, 28
- expressions.cc, 88
- expressions.hpp, 88
- ExprGe, 29
 - eval, 30
 - ExprGe, 30
- ExprGt, 31
 - eval, 31
 - ExprGt, 31
- ExprHart, 33
 - eval, 34
 - ExprHart, 33
- ExprIdent, 34
 - eval, 35
 - ExprIdent, 35
- ExprKnow, 37
 - eval, 38
 - ExprKnow, 37
- ExprLe, 38
 - eval, 39
 - ExprLe, 39
- ExprLt, 40
 - eval, 40
 - ExprLt, 40
- ExprMul, 42
 - eval, 43
 - ExprMul, 42
- ExprNe, 43
 - eval, 44
 - ExprNe, 44
- ExprNode, 45
 - eval, 45
- ExprNot, 46
 - eval, 46
 - ExprNot, 46
- ExprOr, 47
 - eval, 48
 - ExprOr, 47
- ExprRem, 48
 - eval, 49
 - ExprRem, 49
- ExprSub, 50
 - eval, 50
 - ExprSub, 50
- findGlobalStateInEpistemicClass
 - GlobalModelGenerator, 57
- findOrCreateEpistemicClass
 - GlobalModelGenerator, 57
- findOrCreateEpistemicClassForKnowledge
 - GlobalModelGenerator, 58
- Formula, 52
- FormulaTemplate, 52
- generateAgent
 - AgentTemplate, 15
- generateGlobalTransitions
 - GlobalModelGenerator, 58
- generateInitState
 - GlobalModelGenerator, 58
- generateStateFromLocalStates
 - GlobalModelGenerator, 59
- getAgentInstanceByName
 - GlobalModelGenerator, 59
- getCurrentGlobalModel
 - GlobalModelGenerator, 59
- getEpistemicClassForGlobalState
 - Verification, 78
- getFormula
 - GlobalModelGenerator, 60
- getFormulaCorectness
 - GlobalModelGenerator, 60
- getFormulaSize
 - GlobalModelGenerator, 60
- getGlobalStateEnvironment
 - GlobalState, 62
- getLocalStatesSCC
 - Utils.cpp, 96
 - Utils.hpp, 99
- GLOBAL_MODEL
 - DotGraph.hpp, 87
- GlobalModel, 53
- GlobalModel.hpp, 89
- GlobalModelGenerator, 53
 - computeEpistemicClassHash, 55
 - computeGlobalStateHash, 56

- expandAndReduceAllStates, 56
- expandState, 56
- expandStateAndReturn, 56
- findGlobalStateInEpistemicClass, 57
- findOrCreateEpistemicClass, 57
- findOrCreateEpistemicClassForKnowledge, 58
- generateGlobalTransitions, 58
- generateInitState, 58
- generateStateFromLocalStates, 59
- getAgentInstanceByName, 59
- getCurrentGlobalModel, 59
- getFormula, 60
- getFormulaCorectness, 60
- getFormulaSize, 60
- initModel, 60
- GlobalModelGenerator.cpp, 90
- GlobalModelGenerator.hpp, 90
- GlobalModelGenerator::GlobalStateTupleHash, 62
- GlobalState, 61
 - getGlobalStateEnvironment, 62
 - toString, 62
- GlobalState.cpp, 91
- GlobalState.hpp, 91
- GlobalTransition, 63
- GlobalTransition.cpp, 91
- GlobalTransition.hpp, 92
- HistoryDbg, 63
 - addEntry, 64
 - cloneEntry, 64
 - markEntry, 65
 - print, 65
- HistoryEntry, 65
 - toString, 66
- HistoryEntryType
 - Verification.hpp, 103
- includesState
 - Agent, 12
- initModel
 - GlobalModelGenerator, 60
- isAgentInCoalition
 - Verification, 78
- isGlobalTransitionControlledByCoalition
 - Verification, 79
- LOCAL_MODEL
 - DotGraph.hpp, 87
- LocalModels, 67
- localModelsToString
 - Utils.cpp, 97
 - Utils.hpp, 100
- LocalState, 67
 - compare, 68
 - toString, 68
- LocalState.cpp, 92
- LocalState.hpp, 92
- LocalStateTemplate, 69
- LocalTransition, 69
- MARK_DECISION_AS_INVALID
 - Verification.hpp, 103
- markEntry
 - HistoryDbg, 65
- ModelParser, 70
 - parse, 70
- ModelParser.cc, 93
- newHistoryMarkDecisionAsInvalid
 - Verification, 79
- nodes.cc, 93
- nodes.hpp, 94
- NORMAL
 - Verification.hpp, 103
- outputGlobalModel
 - Utils.cpp, 97
 - Utils.hpp, 100
- parse
 - ModelParser, 70
- print
 - HistoryDbg, 65
- printCurrentHistory
 - Verification, 79
- RESTORE
 - Verification.hpp, 103
- restoreHistory
 - Verification, 80
- REVERT
 - Verification.hpp, 103
- revertLastDecision
 - Verification, 80
- saveToFile
 - DotGraph, 21
- setIdent
 - AgentTemplate, 16
- setInitState
 - AgentTemplate, 16
- STATE_STATUS
 - Verification.hpp, 103
- styleString
 - DotGraph, 21
- tarjanVisit
 - Utils.cpp, 97
- TestVerif, 71
- toString
 - GlobalState, 62
 - HistoryEntry, 66
 - LocalState, 68
- TransitionTemplate, 71
 - TransitionTemplate, 72
- TraversalMode
 - Verification.hpp, 103
- Types.hpp, 94
- undoHistoryUntil

- Verification, 81
- undoLastHistoryEntry
 - Verification, 81
- Utils.cpp, 95
 - agentToString, 96
 - envToString, 96
 - getLocalStatesSCC, 96
 - localModelsToString, 97
 - outputGlobalModel, 97
 - tarjanVisit, 97
- Utils.hpp, 98
 - agentToString, 99
 - envToString, 99
 - getLocalStatesSCC, 99
 - localModelsToString, 100
 - outputGlobalModel, 100
- Var, 73
- Verification, 73
 - addHistoryContext, 75
 - addHistoryDecision, 76
 - addHistoryMarkDecisionAsInvalid, 76
 - addHistoryStateStatus, 76
 - areGlobalStatesInTheSameEpistemicClass, 77
 - checkUncontrolledSet, 77
 - equivalentGlobalTransitions, 78
 - getEpistemicClassForGlobalState, 78
 - isAgentInCoalition, 78
 - isGlobalTransitionControlledByCoalition, 79
 - newHistoryMarkDecisionAsInvalid, 79
 - printCurrentHistory, 79
 - restoreHistory, 80
 - revertLastDecision, 80
 - undoHistoryUntil, 81
 - undoLastHistoryEntry, 81
 - Verification, 75
 - verify, 81
 - verifyGlobalState, 81
 - verifyLocalStates, 82
 - verifyTransitionSets, 82
- Verification.cpp, 101
 - dbgHistEnt, 101
 - dbgVerifStatus, 102
 - verStatusToStr, 102
- Verification.hpp, 102
 - CONTEXT, 103
 - DECISION, 103
 - HistoryEntryType, 103
 - MARK_DECISION_AS_INVALID, 103
 - NORMAL, 103
 - RESTORE, 103
 - REVERT, 103
 - STATE_STATUS, 103
 - TraversalMode, 103
 - verStatusToStr, 104
- verify
 - Verification, 81
- verifyGlobalState
 - Verification, 81
- verifyLocalStates
 - Verification, 82
- verifyTransitionSets
 - Verification, 82
- verStatusToStr
 - Verification.cpp, 102
 - Verification.hpp, 104
- yy_bs_column
 - yy_buffer_state, 83
- yy_bs_lineno
 - yy_buffer_state, 83
- yy_buffer_state, 83
 - yy_bs_column, 83
 - yy_bs_lineno, 83
- yy_trans_info, 84
- yyalloc, 84
- YYSTYPE, 84