

stv_v2

Generated by Doxygen 1.9.1

1 STV2 - StraTegic Verifier 2	1
1.1 Usage	1
1.2 Tests	1
1.3 Misc	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	5
3.1 Class List	5
4 File Index	7
4.1 File List	7
5 Class Documentation	9
5.1 Agent Class Reference	9
5.1.1 Detailed Description	9
5.1.2 Constructor & Destructor Documentation	10
5.1.2.1 Agent()	10
5.1.3 Member Function Documentation	10
5.1.3.1 includesState()	10
5.2 AgentTemplate Class Reference	10
5.2.1 Detailed Description	11
5.2.2 Member Function Documentation	11
5.2.2.1 addInitial()	11
5.2.2.2 addLocal()	12
5.2.2.3 addPersistent()	12
5.2.2.4 addTransition()	13
5.2.2.5 generateAgent()	13
5.2.2.6 setIdent()	14
5.2.2.7 setInitState()	15
5.3 Assignment Class Reference	15
5.3.1 Detailed Description	16
5.3.2 Constructor & Destructor Documentation	16
5.3.2.1 Assignment()	16
5.3.3 Member Function Documentation	16
5.3.3.1 assign()	16
5.4 Cfg Struct Reference	16
5.5 Condition Struct Reference	17
5.5.1 Detailed Description	17
5.6 DotGraph Class Reference	17
5.6.1 Member Function Documentation	18
5.6.1.1 addEdge()	18
5.6.1.2 addNode()	18

5.6.1.3 saveToFile()	19
5.6.2 Member Data Documentation	19
5.6.2.1 styleString	19
5.7 EpistemicClass Struct Reference	19
5.7.1 Detailed Description	20
5.8 ExprAdd Class Reference	20
5.8.1 Detailed Description	20
5.8.2 Constructor & Destructor Documentation	20
5.8.2.1 ExprAdd()	20
5.8.3 Member Function Documentation	21
5.8.3.1 eval()	21
5.9 ExprAnd Class Reference	21
5.9.1 Detailed Description	22
5.9.2 Constructor & Destructor Documentation	22
5.9.2.1 ExprAnd()	22
5.9.3 Member Function Documentation	22
5.9.3.1 eval()	22
5.10 ExprConst Class Reference	23
5.10.1 Detailed Description	23
5.10.2 Constructor & Destructor Documentation	23
5.10.2.1 ExprConst()	23
5.10.3 Member Function Documentation	23
5.10.3.1 eval()	23
5.11 ExprDiv Class Reference	24
5.11.1 Detailed Description	24
5.11.2 Constructor & Destructor Documentation	24
5.11.2.1 ExprDiv()	24
5.11.3 Member Function Documentation	25
5.11.3.1 eval()	25
5.12 ExprEq Class Reference	25
5.12.1 Detailed Description	26
5.12.2 Constructor & Destructor Documentation	26
5.12.2.1 ExprEq()	26
5.12.3 Member Function Documentation	26
5.12.3.1 eval()	26
5.13 ExprGe Class Reference	27
5.13.1 Detailed Description	27
5.13.2 Constructor & Destructor Documentation	27
5.13.2.1 ExprGe()	27
5.13.3 Member Function Documentation	27
5.13.3.1 eval()	27
5.14 ExprGt Class Reference	28

5.14.1 Detailed Description	28
5.14.2 Constructor & Destructor Documentation	28
5.14.2.1 ExprGt()	28
5.14.3 Member Function Documentation	29
5.14.3.1 eval()	29
5.15 ExprIdent Class Reference	29
5.15.1 Detailed Description	30
5.15.2 Constructor & Destructor Documentation	30
5.15.2.1 ExprIdent()	30
5.15.3 Member Function Documentation	30
5.15.3.1 eval()	30
5.16 ExprLe Class Reference	31
5.16.1 Detailed Description	31
5.16.2 Constructor & Destructor Documentation	31
5.16.2.1 ExprLe()	31
5.16.3 Member Function Documentation	32
5.16.3.1 eval()	32
5.17 ExprLt Class Reference	32
5.17.1 Detailed Description	32
5.17.2 Constructor & Destructor Documentation	32
5.17.2.1 ExprLt()	32
5.17.3 Member Function Documentation	33
5.17.3.1 eval()	33
5.18 ExprMul Class Reference	33
5.18.1 Detailed Description	34
5.18.2 Constructor & Destructor Documentation	34
5.18.2.1 ExprMul()	34
5.18.3 Member Function Documentation	34
5.18.3.1 eval()	34
5.19 ExprNe Class Reference	35
5.19.1 Detailed Description	35
5.19.2 Constructor & Destructor Documentation	35
5.19.2.1 ExprNe()	35
5.19.3 Member Function Documentation	35
5.19.3.1 eval()	35
5.20 ExprNode Class Reference	36
5.20.1 Detailed Description	36
5.20.2 Member Function Documentation	36
5.20.2.1 eval()	36
5.21 ExprNot Class Reference	37
5.21.1 Detailed Description	37
5.21.2 Constructor & Destructor Documentation	37

5.21.2.1 ExprNot()	37
5.21.3 Member Function Documentation	38
5.21.3.1 eval()	38
5.22 ExprOr Class Reference	38
5.22.1 Detailed Description	38
5.22.2 Constructor & Destructor Documentation	38
5.22.2.1 ExprOr()	38
5.22.3 Member Function Documentation	39
5.22.3.1 eval()	39
5.23 ExprRem Class Reference	39
5.23.1 Detailed Description	40
5.23.2 Constructor & Destructor Documentation	40
5.23.2.1 ExprRem()	40
5.23.3 Member Function Documentation	40
5.23.3.1 eval()	40
5.24 ExprSub Class Reference	41
5.24.1 Detailed Description	41
5.24.2 Constructor & Destructor Documentation	41
5.24.2.1 ExprSub()	41
5.24.3 Member Function Documentation	41
5.24.3.1 eval()	41
5.25 Formula Struct Reference	42
5.26 FormulaTemplate Struct Reference	42
5.26.1 Detailed Description	42
5.27 GlobalModel Struct Reference	43
5.27.1 Detailed Description	43
5.28 GlobalModelGenerator Class Reference	43
5.28.1 Detailed Description	44
5.28.2 Member Function Documentation	44
5.28.2.1 checkLocalTransitionConditions()	44
5.28.2.2 computeEpistemicClassHash()	45
5.28.2.3 computeGlobalStateHash()	45
5.28.2.4 expandState()	46
5.28.2.5 findGlobalStateInEpistemicClass()	46
5.28.2.6 findOrCreateEpistemicClass()	46
5.28.2.7 generateGlobalTransitions()	47
5.28.2.8 generateInitState()	47
5.28.2.9 generateStateFromLocalStates()	47
5.28.2.10 getCurrentGlobalModel()	48
5.28.2.11 getFormula()	48
5.28.2.12 initModel()	48
5.29 GlobalState Struct Reference	49

5.29.1 Detailed Description	49
5.30 GlobalTransition Struct Reference	49
5.30.1 Detailed Description	50
5.31 HistoryDbg Class Reference	50
5.31.1 Detailed Description	51
5.31.2 Member Function Documentation	51
5.31.2.1 addEntry()	51
5.31.2.2 cloneEntry()	51
5.31.2.3 markEntry()	51
5.31.2.4 print()	52
5.32 HistoryEntry Struct Reference	52
5.32.1 Detailed Description	53
5.32.2 Member Function Documentation	53
5.32.2.1 toString()	53
5.33 LocalModels Struct Reference	53
5.33.1 Detailed Description	54
5.34 LocalState Class Reference	54
5.34.1 Detailed Description	54
5.34.2 Member Function Documentation	54
5.34.2.1 compare()	54
5.35 LocalStateTemplate Class Reference	55
5.35.1 Detailed Description	55
5.36 LocalTransition Struct Reference	55
5.36.1 Detailed Description	56
5.37 ModelParser Class Reference	56
5.37.1 Detailed Description	57
5.37.2 Member Function Documentation	57
5.37.2.1 parse()	57
5.38 TransitionTemplate Class Reference	57
5.38.1 Detailed Description	58
5.38.2 Constructor & Destructor Documentation	58
5.38.2.1 TransitionTemplate()	58
5.39 Var Struct Reference	59
5.39.1 Detailed Description	59
5.40 Verification Class Reference	59
5.40.1 Detailed Description	61
5.40.2 Constructor & Destructor Documentation	61
5.40.2.1 Verification()	61
5.40.3 Member Function Documentation	61
5.40.3.1 addHistoryContext()	61
5.40.3.2 addHistoryDecision()	62
5.40.3.3 addHistoryMarkDecisionAsInvalid()	62

5.40.3.4 addHistoryStateStatus()	62
5.40.3.5 areGlobalStatesInTheSameEpistemicClass()	63
5.40.3.6 checkUncontrolledSet()	63
5.40.3.7 equivalentGlobalTransitions()	64
5.40.3.8 getEpistemicClassForGlobalState()	64
5.40.3.9 isAgentInCoalition()	64
5.40.3.10 isGlobalTransitionControlledByCoalition()	65
5.40.3.11 newHistoryMarkDecisionAsInvalid()	65
5.40.3.12 printCurrentHistory()	66
5.40.3.13 restoreHistory()	66
5.40.3.14 revertLastDecision()	66
5.40.3.15 undoHistoryUntil()	67
5.40.3.16 undoLastHistoryEntry()	67
5.40.3.17 verify()	67
5.40.3.18 verifyGlobalState()	68
5.40.3.19 verifyLocalStates()	68
5.40.3.20 verifyTransitionSets()	68
5.41 yy_buffer_state Struct Reference	69
5.41.1 Member Data Documentation	69
5.41.1.1 yy_bs_column	69
5.41.1.2 yy_bs_lineno	70
5.42 yy_trans_info Struct Reference	70
5.43 yyalloc Union Reference	70
5.44 YYSTYPE Union Reference	70
6 File Documentation	71
6.1 expressions.cc File Reference	71
6.1.1 Detailed Description	71
6.2 expressions.hpp File Reference	71
6.2.1 Detailed Description	72
6.3 GlobalModelGenerator.cpp File Reference	72
6.3.1 Detailed Description	72
6.4 GlobalModelGenerator.hpp File Reference	73
6.4.1 Detailed Description	73
6.5 ModelParser.cc File Reference	73
6.5.1 Detailed Description	73
6.6 nodes.cc File Reference	74
6.6.1 Detailed Description	74
6.7 nodes.hpp File Reference	74
6.7.1 Detailed Description	74
6.8 Types.cc File Reference	74
6.8.1 Detailed Description	75

6.9 Types.hpp File Reference	75
6.9.1 Detailed Description	76
6.9.2 Enumeration Type Documentation	76
6.9.2.1 ConditionOperator	76
6.9.2.2 GlobalStateVerificationStatus	76
6.9.2.3 VarAssignmentType	76
6.10 Utils.cpp File Reference	77
6.10.1 Detailed Description	77
6.10.2 Function Documentation	77
6.10.2.1 agentToString()	77
6.10.2.2 envToString()	78
6.10.2.3 getLocalStatesSCC()	78
6.10.2.4 localModelsToString()	78
6.10.2.5 outputGlobalModel()	79
6.10.2.6 tarjanVisit()	79
6.11 Utils.hpp File Reference	80
6.11.1 Function Documentation	80
6.11.1.1 agentToString()	80
6.11.1.2 envToString()	81
6.11.1.3 getLocalStatesSCC()	81
6.11.1.4 localModelsToString()	81
6.11.1.5 outputGlobalModel()	82
6.12 Verification.cpp File Reference	82
6.12.1 Detailed Description	82
6.12.2 Function Documentation	82
6.12.2.1 dbgHistEnt()	83
6.12.2.2 dbgVerifStatus()	83
6.12.2.3 verStatusToStr()	83
6.13 Verification.hpp File Reference	84
6.13.1 Enumeration Type Documentation	84
6.13.1.1 HistoryEntryType	84
6.13.1.2 Mode	85
6.13.2 Function Documentation	85
6.13.2.1 verStatusToStr()	85
Index	87

Chapter 1

STV2 - StraTegic Verifier 2

1.1 Usage

To run:

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

Configuration file:

```
build/config.txt
```

1.2 Tests

To run tests:

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

1.3 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
```

The `dot2png.sh` script converts all `*.dot*` files from a current folder to `*.png*`.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Agent	9
AgentTemplate	10
Assignment	15
Cfg	16
Condition	17
DotGraph	17
EpistemicClass	19
ExprNode	36
ExprAdd	20
ExprAnd	21
ExprConst	23
ExprDiv	24
ExprEq	25
ExprGe	27
ExprGt	28
ExprIdent	29
ExprLe	31
ExprLt	32
ExprMul	33
ExprNe	35
ExprNot	37
ExprOr	38
ExprRem	39
ExprSub	41
Formula	42
FormulaTemplate	42
GlobalModel	43
GlobalModelGenerator	43
GlobalState	49
GlobalTransition	49
HistoryDbg	50
HistoryEntry	52
LocalModels	53
LocalState	54
LocalStateTemplate	55

LocalTransition	55
ModelParser	56
TransitionTemplate	57
Var	59
Verification	59
yy_buffer_state	69
yy_trans_info	70
yyalloc	70
YYSTYPE	70

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	9
AgentTemplate	Represents a single agent loaded from the description from a file	10
Assignment	Represents an assingment	15
Cfg	16
Condition	Represents a condition for LocalTransition	17
DotGraph	17
EpistemicClass	Represents a single epistemic class	19
ExprAdd	Node for addition	20
ExprAnd	Node for AND operator	21
ExprConst	Node for a constant	23
ExprDiv	Node for division	24
ExprEq	Node for "==" operator	25
ExprGe	Node for ">=" operator	27
ExprGt	Node for ">" operator	28
ExprIdent	Node for an identifier	29
ExprLe	Node for "<=" operator	31
ExprLt	Node for "<" operator	32
ExprMul	Node for multiplication	33
ExprNe	Node for "!=" operator	35

ExprNode	Base node for expressions	36
ExprNot	Node for NOT operator	37
ExprOr	Node for OR operator	38
ExprRem	Node for modulo	39
ExprSub	Node for subtraction	41
Formula	42
FormulaTemplate	Contains a template for coalition of Agent as string from the formula	42
GlobalModel	Represents a global model, containing agents and a formula	43
GlobalModelGenerator	Stores the local models, formula and a global model	43
GlobalState	Represents a single global state	49
GlobalTransition	Represents a single global transition	49
HistoryDbg	Stores history and allows displaying it to the console	50
HistoryEntry	Structure used to save model traversal history	52
LocalModels	Represents a single local model, contains all agents and variables	53
LocalState	Represents a single LocalState , containing id, name and internal variables	54
LocalStateTemplate	A template for the local state	55
LocalTransition	Represents a single local transition, containing id, global name, local name, is shared and count of the appearances	55
ModelParser	A parser for converting a text file into a model	56
TransitionTemplate	Represents a meta-transition	57
Var	Represents a variable in the model, containing name, initial value and persistence	59
Verification	A class that verifies if the model fulfills the formula. Also can do some operations on decision history	59
yy_buffer_state	69
yy_trans_info	70
yyalloc	70
YYSTYPE	70

Chapter 4

File Index

4.1 File List

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

clock_test.h	??
config.h	??
Constants.hpp	??
DotGraph.hpp	??
expressions.cc	
Eval and helper class for expressions. Eval and helper class for expressions	71
expressions.hpp	
Eval and helper class for expressions. Eval and helper class for expressions	71
general_test.h	??
GlobalModelGenerator.cpp	
Generator of a global model. Class for initializing and generating a global model	72
GlobalModelGenerator.hpp	
Generator of a global model. Class for initializing and generating a global model	73
ModelParser.cc	
A model parser. A parser for converting a text file into a model	73
ModelParser.hpp	??
nodes.cc	
Parser templates. Class for setting up a new objects from a parser	74
nodes.hpp	
Parser templates. Class for setting up a new objects from a parser	74
parser.h	??
simple_voting_run_test.h	??
simple_voting_test.h	??
simple_voting_with_fakes_test.h	??
test_test.h	??
trains_test.h	??
trains_with_bridge_test.h	??
Types.cc	
Custom data structures. Data structures and classes containing model data	74
Types.hpp	
Custom data structures. Data structures and classes containing model data	75
Utils.cpp	
Utility functions. A collection of utility functions to use in the project	77
Utils.hpp	80
Verification.cpp	
Class for verification of the formula on a model. Class for verification of the specified formula on a specified model	82
Verification.hpp	84

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 <Types.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:

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

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

- `char * fname`

- char **stv_mode**
- bool **output_local_models**
- bool **output_global_model**
- bool **output_dot_files**
- int **model_id**

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

- Constants.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)
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 filename="")
creates a .dot file

Public Attributes

- `std::vector< std::string > nodes`
- `std::vector< std::string > edges`
- `std::string name`
- `std::string caption`

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 filename = "" )
```

creates a .dot file

Parameters

<i>filename</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=\"#ffffff\",\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 <Types.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:

- [Types.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)
Calculates the expression value.

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

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

5.8.3 Member Function Documentation

5.8.3.1 eval()

```
int ExprAdd::eval (
    Environment & env ) [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](#)

5.9 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)
Calculates the expression value.

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

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

5.9.3 Member Function Documentation

5.9.3.1 eval()

```
int ExprAnd::eval (
    Environment & env ) [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](#)

5.10 ExprConst Class Reference

Node for a constant.

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

Public Member Functions

- [ExprConst](#) (int _val)
Constant expression constructor.
- virtual int [eval](#) ([Environment](#) &env)
Calculates the expression value.

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 ) [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`)
Calculates the expression value.

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

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

5.11.3 Member Function Documentation

5.11.3.1 eval()

```
int ExprDiv::eval (  
    Environment & env ) [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](#)

5.12 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)
Calculates the expression value.

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

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

5.12.3 Member Function Documentation

5.12.3.1 eval()

```
int ExprEq::eval (
    Environment & env ) [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](#)

5.13 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)
Calculates the expression value.

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

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

5.13.3 Member Function Documentation

5.13.3.1 eval()

```
int ExprGe::eval (
    Environment & env ) [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)
Calculates the expression value.

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

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

5.14.3 Member Function Documentation

5.14.3.1 eval()

```
int ExprGt::eval (
    Environment & env ) [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](#)

5.15 ExprIdent Class Reference

Node for an identifier.

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

Public Member Functions

- [ExprIdent](#) (string _ident)
Identifier expression constructor.
- virtual int [eval](#) ([Environment](#) &env)
Calculates the expression value.

5.15.1 Detailed Description

Node for an identifier.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 ExprIdent()

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

Identifier expression constructor.

Parameters

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

5.15.3 Member Function Documentation

5.15.3.1 eval()

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

Calculates the expression value.

Parameters

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

Returns

Returns an integer.

Parameters

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

Returns

Implements [ExprNode](#).

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

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

5.16 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)
Calculates the expression value.

5.16.1 Detailed Description

Node for "<=" operator.

5.16.2 Constructor & Destructor Documentation

5.16.2.1 ExprLe()

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

Less or equal expression constructor.

Parameters

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

5.16.3 Member Function Documentation

5.16.3.1 eval()

```
int ExprLe::eval (
    Environment & env ) [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](#)

5.17 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)
Calculates the expression value.

5.17.1 Detailed Description

Node for "<" operator.

5.17.2 Constructor & Destructor Documentation

5.17.2.1 ExprLt()

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

Less than expression constructor.

Parameters

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

5.17.3 Member Function Documentation

5.17.3.1 eval()

```
int ExprLt::eval (
    Environment & env ) [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 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)
Calculates the expression value.

5.18.1 Detailed Description

Node for multiplication.

5.18.2 Constructor & Destructor Documentation

5.18.2.1 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.

5.18.3 Member Function Documentation

5.18.3.1 eval()

```
int ExprMul::eval (
    Environment & env ) [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](#)

5.19 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)
Calculates the expression value.

5.19.1 Detailed Description

Node for "!=" operator.

5.19.2 Constructor & Destructor Documentation

5.19.2.1 ExprNe()

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

Not equals 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 ExprNe::eval (
    Environment & env ) [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 ExprNode Class Reference

Base node for expressions.

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

Public Member Functions

- virtual int [eval](#) ([Environment](#) &env)=0
Calculates the expression value.

5.20.1 Detailed Description

Base node for expressions.

5.20.2 Member Function Documentation

5.20.2.1 eval()

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

Calculates the expression value.

Parameters

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

Returns

Returns an integer.

Implemented in [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](#)

5.21 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)
Calculates the expression value.

5.21.1 Detailed Description

Node for NOT operator.

5.21.2 Constructor & Destructor Documentation

5.21.2.1 ExprNot()

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

Logic NOT expression constructor.

Parameters

<code>_arg</code>	Calculates the expression value.
-------------------	----------------------------------

5.21.3 Member Function Documentation

5.21.3.1 eval()

```
int ExprNot::eval (
    Environment & env ) [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 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)
Calculates the expression value.

5.22.1 Detailed Description

Node for OR operator.

5.22.2 Constructor & Destructor Documentation

5.22.2.1 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.

5.22.3 Member Function Documentation

5.22.3.1 eval()

```
int ExprOr::eval (
    Environment & env ) [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](#)

5.23 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)
Calculates the expression value.

5.23.1 Detailed Description

Node for modulo.

5.23.2 Constructor & Destructor Documentation

5.23.2.1 ExprRem()

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

Modulo expression constructor.

Parameters

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

5.23.3 Member Function Documentation

5.23.3.1 eval()

```
int ExprRem::eval (
    Environment & env ) [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](#)

5.24 ExprSub Class Reference

Node for subtraction.

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

Public Member Functions

- [ExprSub](#) ([ExprNode](#) **_l*arg, [ExprNode](#) **_r*arg)
Subtraction expression constructor.
- virtual int [eval](#) ([Environment](#) &env)
Calculates the expression value.

5.24.1 Detailed Description

Node for subtraction.

5.24.2 Constructor & Destructor Documentation

5.24.2.1 ExprSub()

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

Subtraction expression constructor.

Parameters

<i>_l</i> arg	Left argument of the expression.
<i>_r</i> arg	Right argument of the expression.

5.24.3 Member Function Documentation

5.24.3.1 eval()

```
int ExprSub::eval (
    Environment & env ) [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](#)

5.25 Formula Struct Reference

Public Attributes

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

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

- [Types.hpp](#)

5.26 FormulaTemplate Struct Reference

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

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

Public Attributes

- `set< string > * coalition`
- `ExprNode * formula`

5.26.1 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](#)

5.27 GlobalModel Struct Reference

Represents a global model, containing agents and a formula.

```
#include <Types.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.
- `vector< GlobalTransition * > globalTransitions`
Every [GlobalTransition](#) in the model.
- `map< Agent *, map< string, EpistemicClass * > > epistemicClasses`
Map of [Agent](#) pointers to a map of [EpistemicClass](#).

5.27.1 Detailed Description

Represents a global model, containing agents and a formula.

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

- [Types.hpp](#)

5.28 GlobalModelGenerator Class Reference

Stores the local models, formula and a global model.

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

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.
- `void expandAllStates ()`
Expands the states starting from the initial [GlobalState](#) 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.

Protected Member Functions

- `GlobalState * generateInitState ()`
Generates initial state of the model from `GlobalModel` in memory.
- `GlobalState * generateStateFromLocalStates (set< 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`.
- `bool checkLocalTransitionConditions (LocalTransition *localTransition, GlobalState *globalState)`
Checks if all conditions for a given local transition in a given global state are fulfilled.
- `string computeEpistemicClassHash (set< LocalState * > *localStates, Agent *agent)`
Creates a hash from a set of `LocalState` and an `Agent`.
- `string computeGlobalStateHash (set< LocalState * > *localStates)`
Creates a hash from a set of `LocalState`.
- `EpistemicClass * findOrCreateEpistemicClass (set< LocalState * > *localStates, Agent *agent)`
Checks if a set of `LocalState` is already an epistemic class for a given `Agent`, if not, creates a new one.
- `GlobalState * findGlobalStateInEpistemicClass (set< 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`.

5.28.1 Detailed Description

Stores the local models, formula and a global model.

5.28.2 Member Function Documentation

5.28.2.1 checkLocalTransitionConditions()

```
bool GlobalModelGenerator::checkLocalTransitionConditions (
    LocalTransition * localTransition,
    GlobalState * globalState ) [protected]
```

Checks if all conditions for a given local transition in a given global state are fulfilled.

Parameters

<i>localTransition</i>	Local transition to traverse.
<i>globalState</i>	Current global state.

Returns

Returns true if all of the necessary conditions to use that transition are fulfilled, otherwise false.

5.28.2.2 computeEpistemicClassHash()

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

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

Parameters

<i>localStates</i>	Pointer to a set of pointers of LocalState and pointer to and Agent to turn into a hash.
--------------------	--

Returns

Returns a string with a hash.

5.28.2.3 computeGlobalStateHash()

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

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

Parameters

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

Returns

Returns a string with a hash.

5.28.2.4 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.
--------------	--

5.28.2.5 findGlobalStateInEpistemicClass()

```
GlobalState * GlobalModelGenerator::findGlobalStateInEpistemicClass (
    set< 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 set 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.

5.28.2.6 findOrCreateEpistemicClass()

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

Checks if a set 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](#).

5.28.2.7 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.

5.28.2.8 generateInitState()

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

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

Returns

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

5.28.2.9 generateStateFromLocalStates()

```
GlobalState * GlobalModelGenerator::generateStateFromLocalStates (
    set< 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](#).

5.28.2.10 `getCurrentGlobalModel()`

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

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

Returns

Returns a pointer to a global model.

5.28.2.11 `getFormula()`

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

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

Returns

Returns a pointer to the formula structure.

5.28.2.12 `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](#)

5.29 GlobalState Struct Reference

Represents a single global state.

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

Public Attributes

- `int id`
Identifier of the global state.
- `string hash`
Hash of the global state used in quick checks if the states are in the same epistemic class.
- `map< Var *, int > vars`
Map of model variables and their current values.
- `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.
- `set< LocalState * > localStates`
Local states of each agent that define this global state.

5.29.1 Detailed Description

Represents a single global state.

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

- [Types.hpp](#)

5.30 GlobalTransition Struct Reference

Represents a single global transition.

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

Public Attributes

- `int 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.

5.30.1 Detailed Description

Represents a single global transition.

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

- [Types.hpp](#)

5.31 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.

5.31.1 Detailed Description

Stores history and allows displaying it to the console.

5.31.2 Member Function Documentation

5.31.2.1 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.
--------------	--

5.31.2.2 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.

5.31.2.3 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 .

5.31.2.4 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](#)

5.32 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](#).

5.32.1 Detailed Description

Structure used to save model traversal history.

5.32.2 Member Function Documentation

5.32.2.1 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](#)

5.33 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.

5.33.1 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](#)

5.34 LocalState Class Reference

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

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

Public Member Functions

- `bool compare (LocalState *state)`
Function comparing two states.

Public Attributes

- `int 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](#).

5.34.1 Detailed Description

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

5.34.2 Member Function Documentation

5.34.2.1 `compare()`

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

Function comparing two states.

Parameters

<code>state</code>	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.

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

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

5.35 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.

5.35.1 Detailed Description

A template for the local state.

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

- [nodes.hpp](#)

5.36 LocalTransition Struct Reference

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

```
#include <Types.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.
- `set< LocalTransition * > sharedLocalTransitions`
Stores shared transitions from different models.

5.36.1 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:

- [Types.hpp](#)

5.37 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.

5.37.1 Detailed Description

A parser for converting a text file into a model.

5.37.2 Member Function Documentation

5.37.2.1 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](#)

5.38 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.

5.38.1 Detailed Description

Represents a meta-transition.

5.38.2 Constructor & Destructor Documentation

5.38.2.1 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](#)

5.39 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.

5.39.1 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](#)

5.40 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.

Protected Member Functions

- bool `verifyLocalStates` (set< `LocalState` * > *localStates)
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 global↔
TransitionControlled)
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` *global↔
State, 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)
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

- [Mode](#) `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.

5.40.1 Detailed Description

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

5.40.2 Constructor & Destructor Documentation

5.40.2.1 Verification()

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

Constructor for [Verification](#).

Parameters

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

5.40.3 Member Function Documentation

5.40.3.1 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.

5.40.3.2 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.

5.40.3.3 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.

5.40.3.4 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.

5.40.3.5 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](#).

5.40.3.6 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.

5.40.3.7 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.

5.40.3.8 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.

5.40.3.9 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.

5.40.3.10 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.

5.40.3.11 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](#).

5.40.3.12 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.
--------------	--

5.40.3.13 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

5.40.3.14 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.

5.40.3.15 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.

5.40.3.16 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.
-------------------	--

5.40.3.17 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.

5.40.3.18 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.

5.40.3.19 verifyLocalStates()

```
bool Verification::verifyLocalStates (
    set< LocalState * > * localStates ) [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.

5.40.3.20 verifyTransitionSets()

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

5.41 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**

5.41.1 Member Data Documentation

5.41.1.1 yy_bs_column

```
int yy_buffer_state::yy_bs_column
```

The column count.

5.41.1.2 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

5.42 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

5.43 yyalloc 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

5.44 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**

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

- parser.h

Chapter 6

File Documentation

6.1 expressions.cc File Reference

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

```
#include "expressions.hpp"
```

6.1.1 Detailed Description

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

6.2 expressions.hpp File Reference

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

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

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.*

Typedefs

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

6.2.1 Detailed Description

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

6.3 GlobalModelGenerator.cpp File Reference

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

```
#include "GlobalModelGenerator.hpp"
#include <algorithm>
#include <iostream>
```

6.3.1 Detailed Description

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

6.4 GlobalModelGenerator.hpp File Reference

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

```
#include "Constants.hpp"
#include "Types.hpp"
```

Classes

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

6.4.1 Detailed Description

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

6.5 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 <tuple>
```

Functions

- int **yyparse** ()
- void **yyrestart** (FILE *)

Variables

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

6.5.1 Detailed Description

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

6.6 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>
```

6.6.1 Detailed Description

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

6.7 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.

6.7.1 Detailed Description

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

6.8 Types.cc File Reference

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

```
#include "Types.hpp"
```

6.8.1 Detailed Description

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

6.9 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 "reader/expressions.hpp"
```

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](#)
- class [Agent](#)
Contains all data for a single [Agent](#), including id, name and all of the agents' variables.
- class [LocalState](#)
Represents a single [LocalState](#), containing id, name and internal variables.
- struct [LocalTransition](#)
Represents a single local transition, containing id, global name, local name, is shared and count of the appearances.
- struct [LocalModels](#)
Represents a single local model, contains all agents and variables.
- struct [GlobalModel](#)
Represents a global model, containing agents and a formula.
- struct [GlobalState](#)
Represents a single global state.
- struct [GlobalTransition](#)
Represents a single global transition.
- struct [EpistemicClass](#)
Represents a single epistemic class.

Enumerations

- enum [ConditionOperator](#) { [Equals](#) , [NotEquals](#) }
Conditional operator for the variable.
- enum [GlobalStateVerificationStatus](#) { [UNVERIFIED](#) , [PENDING](#) , [VERIFIED_OK](#) , [VERIFIED_ERR](#) }
Verification status of a [GlobalState](#).
- enum [VarAssignmentType](#) { [FromVar](#) , [FromValue](#) }
Handles if the [Var](#) value is from srcVar or from value.

6.9.1 Detailed Description

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

6.9.2 Enumeration Type Documentation

6.9.2.1 ConditionOperator

enum `ConditionOperator`

Conditional operator for the variable.

Enumerator

Equals	Variable should be equal to the value.
NotEquals	Variable should be not equal to the value.

6.9.2.2 GlobalStateVerificationStatus

enum `GlobalStateVerificationStatus`

Verification status of a `GlobalState`.

Enumerator

UNVERIFIED	State is not verified.
PENDING	Entered the state but it is not verified as correct or incorrect yet.
VERIFIED_OK	The state has been verified and is correct.

6.9.2.3 VarAssignmentType

enum `VarAssignmentType`

Handles if the `Var` value is from srcVar or from value.

Enumerator

FromVar	Take value from srcVar.
FromValue	Take value from value.

6.10 Utils.cpp File Reference

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

```
#include "Utils.hpp"
```

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 [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.
- vector< set< [LocalState](#) * > > [getLocalStatesSCC](#) ([Agent](#) *agt)
a quick implementation of a Tarjan SCC algorithm (based on DFS)

Variables

- [Cfg](#) config

6.10.1 Detailed Description

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

6.10.2 Function Documentation

6.10.2.1 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.

6.10.2.2 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)"

6.10.2.3 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

6.10.2.4 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.

6.10.2.5 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.
--------------------	---

6.10.2.6 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 dindex 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

6.11 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)

6.11.1 Function Documentation

6.11.1.1 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.

6.11.1.2 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)"

6.11.1.3 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

6.11.1.4 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.

6.11.1.5 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.
--------------------	---

6.12 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"
```

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

6.12.1 Detailed Description

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

6.12.2 Function Documentation

6.12.2.1 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 HistoryEntry struct which will be printed out.

6.12.2.2 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".

6.12.2.3 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.

6.13 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 [Mode](#) { [NORMAL](#) , [REVERT](#) , [RESTORE](#) }
Current model traversal mode.

Functions

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

6.13.1 Enumeration Type Documentation

6.13.1.1 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.

6.13.1.2 Mode

enum [Mode](#)

Current model traversal mode.

Enumerator

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

6.13.2 Function Documentation

6.13.2.1 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, [18](#)
- addEntry
 - HistoryDbg, [51](#)
- addHistoryContext
 - Verification, [61](#)
- addHistoryDecision
 - Verification, [62](#)
- addHistoryMarkDecisionAsInvalid
 - Verification, [62](#)
- addHistoryStateStatus
 - Verification, [62](#)
- addInitial
 - AgentTemplate, [11](#)
- addLocal
 - AgentTemplate, [12](#)
- addNode
 - DotGraph, [18](#)
- addPersistent
 - AgentTemplate, [12](#)
- addTransition
 - AgentTemplate, [13](#)
- Agent, [9](#)
 - Agent, [10](#)
 - includesState, [10](#)
- AgentTemplate, [10](#)
 - addInitial, [11](#)
 - addLocal, [12](#)
 - addPersistent, [12](#)
 - addTransition, [13](#)
 - generateAgent, [13](#)
 - setIdent, [14](#)
 - setInitState, [14](#)
- agentToString
 - Utils.cpp, [77](#)
 - Utils.hpp, [80](#)
- areGlobalStatesInTheSameEpistemicClass
 - Verification, [63](#)
- assign
 - Assignment, [16](#)
- Assignment, [15](#)
 - assign, [16](#)
 - Assignment, [16](#)
- Cfg, [16](#)
- checkLocalTransitionConditions
 - GlobalModelGenerator, [44](#)
- checkUncontrolledSet
 - Verification, [63](#)
- cloneEntry
 - HistoryDbg, [51](#)
- compare
 - LocalState, [54](#)
- computeEpistemicClassHash
 - GlobalModelGenerator, [45](#)
- computeGlobalStateHash
 - GlobalModelGenerator, [45](#)
- Condition, [17](#)
- ConditionOperator
 - Types.hpp, [76](#)
- CONTEXT
 - Verification.hpp, [84](#)
- dbgHistEnt
 - Verification.cpp, [82](#)
- dbgVerifStatus
 - Verification.cpp, [83](#)
- DECISION
 - Verification.hpp, [84](#)
- DotGraph, [17](#)
 - addEdge, [18](#)
 - addNode, [18](#)
 - saveToFile, [19](#)
 - styleString, [19](#)
- envToString
 - Utils.cpp, [78](#)
 - Utils.hpp, [80](#)
- EpistemicClass, [19](#)
- Equals
 - Types.hpp, [76](#)
- equivalentGlobalTransitions
 - Verification, [64](#)
- eval
 - ExprAdd, [21](#)
 - ExprAnd, [22](#)
 - ExprConst, [23](#)
 - ExprDiv, [25](#)
 - ExprEq, [26](#)
 - ExprGe, [27](#)
 - ExprGt, [29](#)
 - ExprIdent, [30](#)
 - ExprLe, [32](#)
 - ExprLt, [33](#)
 - ExprMul, [34](#)
 - ExprNe, [35](#)
 - ExprNode, [36](#)
 - ExprNot, [38](#)
 - ExprOr, [39](#)
 - ExprRem, [40](#)

- ExprSub, 41
- expandState
 - GlobalModelGenerator, 45
- ExprAdd, 20
 - eval, 21
 - ExprAdd, 20
- ExprAnd, 21
 - eval, 22
 - ExprAnd, 22
- ExprConst, 23
 - eval, 23
 - ExprConst, 23
- ExprDiv, 24
 - eval, 25
 - ExprDiv, 24
- ExprEq, 25
 - eval, 26
 - ExprEq, 26
- expressions.cc, 71
- expressions.hpp, 71
- ExprGe, 27
 - eval, 27
 - ExprGe, 27
- ExprGt, 28
 - eval, 29
 - ExprGt, 28
- ExprIdent, 29
 - eval, 30
 - ExprIdent, 30
- ExprLe, 31
 - eval, 32
 - ExprLe, 31
- ExprLt, 32
 - eval, 33
 - ExprLt, 32
- ExprMul, 33
 - eval, 34
 - ExprMul, 34
- ExprNe, 35
 - eval, 35
 - ExprNe, 35
- ExprNode, 36
 - eval, 36
- ExprNot, 37
 - eval, 38
 - ExprNot, 37
- ExprOr, 38
 - eval, 39
 - ExprOr, 38
- ExprRem, 39
 - eval, 40
 - ExprRem, 40
- ExprSub, 41
 - eval, 41
 - ExprSub, 41
- findGlobalStateInEpistemicClass
 - GlobalModelGenerator, 46
- findOrCreateEpistemicClass
 - GlobalModelGenerator, 46
- Formula, 42
- FormulaTemplate, 42
- FromValue
 - Types.hpp, 76
- FromVar
 - Types.hpp, 76
- generateAgent
 - AgentTemplate, 13
- generateGlobalTransitions
 - GlobalModelGenerator, 47
- generateInitState
 - GlobalModelGenerator, 47
- generateStateFromLocalStates
 - GlobalModelGenerator, 47
- getCurrentGlobalModel
 - GlobalModelGenerator, 48
- getEpistemicClassForGlobalState
 - Verification, 64
- getFormula
 - GlobalModelGenerator, 48
- getLocalStatesSCC
 - Utils.cpp, 78
 - Utils.hpp, 81
- GlobalModel, 43
- GlobalModelGenerator, 43
 - checkLocalTransitionConditions, 44
 - computeEpistemicClassHash, 45
 - computeGlobalStateHash, 45
 - expandState, 45
 - findGlobalStateInEpistemicClass, 46
 - findOrCreateEpistemicClass, 46
 - generateGlobalTransitions, 47
 - generateInitState, 47
 - generateStateFromLocalStates, 47
 - getCurrentGlobalModel, 48
 - getFormula, 48
 - initModel, 48
- GlobalModelGenerator.cpp, 72
- GlobalModelGenerator.hpp, 73
- GlobalState, 49
- GlobalStateVerificationStatus
 - Types.hpp, 76
- GlobalTransition, 49
- HistoryDbg, 50
 - addEntry, 51
 - cloneEntry, 51
 - markEntry, 51
 - print, 52
- HistoryEntry, 52
 - toString, 53
- HistoryEntryType
 - Verification.hpp, 84
- includesState
 - Agent, 10
- initModel

- GlobalModelGenerator, [48](#)
- isAgentInCoalition
 - Verification, [64](#)
- isGlobalTransitionControlledByCoalition
 - Verification, [65](#)
- LocalModels, [53](#)
- localModelsToString
 - Utils.cpp, [78](#)
 - Utils.hpp, [81](#)
- LocalState, [54](#)
 - compare, [54](#)
- LocalStateTemplate, [55](#)
- LocalTransition, [55](#)
- MARK_DECISION_AS_INVALID
 - Verification.hpp, [84](#)
- markEntry
 - HistoryDbg, [51](#)
- Mode
 - Verification.hpp, [85](#)
- ModelParser, [56](#)
 - parse, [57](#)
- ModelParser.cc, [73](#)
- newHistoryMarkDecisionAsInvalid
 - Verification, [65](#)
- nodes.cc, [74](#)
- nodes.hpp, [74](#)
- NORMAL
 - Verification.hpp, [85](#)
- NotEquals
 - Types.hpp, [76](#)
- outputGlobalModel
 - Utils.cpp, [79](#)
 - Utils.hpp, [82](#)
- parse
 - ModelParser, [57](#)
- PENDING
 - Types.hpp, [76](#)
- print
 - HistoryDbg, [52](#)
- printCurrentHistory
 - Verification, [65](#)
- RESTORE
 - Verification.hpp, [85](#)
- restoreHistory
 - Verification, [66](#)
- REVERT
 - Verification.hpp, [85](#)
- revertLastDecision
 - Verification, [66](#)
- saveToFile
 - DotGraph, [19](#)
- setIdent
 - AgentTemplate, [14](#)
- setInitState
 - AgentTemplate, [14](#)
- STATE_STATUS
 - Verification.hpp, [84](#)
- styleString
 - DotGraph, [19](#)
- tarjanVisit
 - Utils.cpp, [79](#)
- toString
 - HistoryEntry, [53](#)
- TransitionTemplate, [57](#)
 - TransitionTemplate, [58](#)
- Types.cc, [74](#)
- Types.hpp, [75](#)
 - ConditionOperator, [76](#)
 - Equals, [76](#)
 - FromValue, [76](#)
 - FromVar, [76](#)
 - GlobalStateVerificationStatus, [76](#)
 - NotEquals, [76](#)
 - PENDING, [76](#)
 - UNVERIFIED, [76](#)
 - VarAssignmentType, [76](#)
 - VERIFIED_OK, [76](#)
- undoHistoryUntil
 - Verification, [67](#)
- undoLastHistoryEntry
 - Verification, [67](#)
- UNVERIFIED
 - Types.hpp, [76](#)
- Utils.cpp, [77](#)
 - agentToString, [77](#)
 - envToString, [78](#)
 - getLocalStatesSCC, [78](#)
 - localModelsToString, [78](#)
 - outputGlobalModel, [79](#)
 - tarjanVisit, [79](#)
- Utils.hpp, [80](#)
 - agentToString, [80](#)
 - envToString, [80](#)
 - getLocalStatesSCC, [81](#)
 - localModelsToString, [81](#)
 - outputGlobalModel, [82](#)
- Var, [59](#)
- VarAssignmentType
 - Types.hpp, [76](#)
- Verification, [59](#)
 - addHistoryContext, [61](#)
 - addHistoryDecision, [62](#)
 - addHistoryMarkDecisionAsInvalid, [62](#)
 - addHistoryStateStatus, [62](#)
 - areGlobalStatesInTheSameEpistemicClass, [63](#)
 - checkUncontrolledSet, [63](#)
 - equivalentGlobalTransitions, [64](#)
 - getEpistemicClassForGlobalState, [64](#)
 - isAgentInCoalition, [64](#)

- isGlobalTransitionControlledByCoalition, 65
- newHistoryMarkDecisionAsInvalid, 65
- printCurrentHistory, 65
- restoreHistory, 66
- revertLastDecision, 66
- undoHistoryUntil, 67
- undoLastHistoryEntry, 67
- Verification, 61
- verify, 67
- verifyGlobalState, 67
- verifyLocalStates, 68
- verifyTransitionSets, 68
- Verification.cpp, 82
 - dbgHistEnt, 82
 - dbgVerifStatus, 83
 - verStatusToStr, 83
- Verification.hpp, 84
 - CONTEXT, 84
 - DECISION, 84
 - HistoryEntryType, 84
 - MARK_DECISION_AS_INVALID, 84
 - Mode, 85
 - NORMAL, 85
 - RESTORE, 85
 - REVERT, 85
 - STATE_STATUS, 84
 - verStatusToStr, 85
- VERIFIED_OK
 - Types.hpp, 76
- verify
 - Verification, 67
- verifyGlobalState
 - Verification, 67
- verifyLocalStates
 - Verification, 68
- verifyTransitionSets
 - Verification, 68
- verStatusToStr
 - Verification.cpp, 83
 - Verification.hpp, 85
- yy_bs_column
 - yy_buffer_state, 69
- yy_bs_lineno
 - yy_buffer_state, 69
- yy_buffer_state, 69
 - yy_bs_column, 69
 - yy_bs_lineno, 69
- yy_trans_info, 70
- yyalloc, 70
- YYSTYPE, 70