

stv\_v2

Generated by Doxygen 1.9.6



<b>1 README</b>	<b>1</b>
<b>2 Hierarchical Index</b>	<b>3</b>
2.1 Class Hierarchy	3
<b>3 Class Index</b>	<b>5</b>
3.1 Class List	5
<b>4 File Index</b>	<b>7</b>
4.1 File List	7
<b>5 Class Documentation</b>	<b>9</b>
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 Member Function Documentation	11
5.2.1.1 addInitial()	11
5.2.1.2 addLocal()	11
5.2.1.3 addPersistent()	12
5.2.1.4 addTransition()	12
5.2.1.5 generateAgent()	12
5.2.1.6 setIdent()	13
5.2.1.7 setInitState()	13
5.3 Assignment Class Reference	14
5.3.1 Detailed Description	14
5.3.2 Constructor & Destructor Documentation	14
5.3.2.1 Assignment()	14
5.4 Cfg Struct Reference	15
5.5 Condition Struct Reference	15
5.5.1 Detailed Description	15
5.6 EpistemicClass Struct Reference	15
5.6.1 Detailed Description	16
5.7 ExprAdd Class Reference	16
5.7.1 Detailed Description	16
5.7.2 Member Function Documentation	16
5.7.2.1 eval()	16
5.8 ExprAnd Class Reference	17
5.8.1 Detailed Description	17
5.8.2 Member Function Documentation	17
5.8.2.1 eval()	17

5.9 ExprConst Class Reference	17
5.9.1 Detailed Description	18
5.9.2 Member Function Documentation	18
5.9.2.1 eval()	18
5.10 ExprDiv Class Reference	18
5.10.1 Detailed Description	18
5.10.2 Member Function Documentation	18
5.10.2.1 eval()	19
5.11 ExprEq Class Reference	19
5.11.1 Detailed Description	19
5.11.2 Member Function Documentation	19
5.11.2.1 eval()	19
5.12 ExprGe Class Reference	20
5.12.1 Detailed Description	20
5.12.2 Member Function Documentation	20
5.12.2.1 eval()	20
5.13 ExprGt Class Reference	20
5.13.1 Detailed Description	21
5.13.2 Member Function Documentation	21
5.13.2.1 eval()	21
5.14 ExprIdent Class Reference	21
5.14.1 Detailed Description	21
5.14.2 Member Function Documentation	21
5.14.2.1 eval()	22
5.15 ExprLe Class Reference	22
5.15.1 Detailed Description	22
5.15.2 Member Function Documentation	22
5.15.2.1 eval()	22
5.16 ExprLt Class Reference	23
5.16.1 Detailed Description	23
5.16.2 Member Function Documentation	23
5.16.2.1 eval()	23
5.17 ExprMul Class Reference	23
5.17.1 Detailed Description	24
5.17.2 Member Function Documentation	24
5.17.2.1 eval()	24
5.18 ExprNe Class Reference	24
5.18.1 Detailed Description	24
5.18.2 Member Function Documentation	24
5.18.2.1 eval()	25
5.19 ExprNode Class Reference	25
5.19.1 Detailed Description	25

5.20 ExprNot Class Reference	25
5.20.1 Detailed Description	26
5.20.2 Member Function Documentation	26
5.20.2.1 eval()	26
5.21 ExprOr Class Reference	26
5.21.1 Detailed Description	26
5.21.2 Member Function Documentation	26
5.21.2.1 eval()	27
5.22 ExprRem Class Reference	27
5.22.1 Detailed Description	27
5.22.2 Member Function Documentation	27
5.22.2.1 eval()	27
5.23 ExprSub Class Reference	28
5.23.1 Detailed Description	28
5.23.2 Member Function Documentation	28
5.23.2.1 eval()	28
5.24 Formula Struct Reference	28
5.24.1 Detailed Description	29
5.25 GlobalModel Struct Reference	29
5.25.1 Detailed Description	29
5.26 GlobalModelGenerator Class Reference	29
5.26.1 Member Function Documentation	30
5.26.1.1 checkLocalTransitionConditions()	30
5.26.1.2 computeEpistemicClassHash()	31
5.26.1.3 computeGlobalStateHash()	31
5.26.1.4 expandState()	31
5.26.1.5 findGlobalStateInEpistemicClass()	32
5.26.1.6 findOrCreateEpistemicClass()	32
5.26.1.7 generateGlobalTransitions()	32
5.26.1.8 generateInitState()	33
5.26.1.9 generateStateFromLocalStates()	33
5.26.1.10 getCurrentGlobalModel()	33
5.26.1.11 getFormula()	34
5.26.1.12 initModel()	34
5.27 GlobalState Struct Reference	34
5.27.1 Detailed Description	35
5.28 GlobalTransition Struct Reference	35
5.28.1 Detailed Description	36
5.29 HistoryDbg Class Reference	36
5.29.1 Member Function Documentation	36
5.29.1.1 addEntry()	36
5.29.1.2 cloneEntry()	37

5.29.1.3 markEntry()	37
5.29.1.4 print()	37
5.30 HistoryEntry Struct Reference	38
5.30.1 Detailed Description	38
5.30.2 Member Function Documentation	38
5.30.2.1 toString()	39
5.31 LocalModels Struct Reference	39
5.31.1 Detailed Description	39
5.32 LocalState Class Reference	39
5.32.1 Detailed Description	40
5.32.2 Member Function Documentation	40
5.32.2.1 compare()	40
5.33 LocalStateTemplate Class Reference	40
5.34 LocalTransition Struct Reference	41
5.34.1 Detailed Description	41
5.35 SeleneFormula Class Reference	42
5.36 SeleneFormula1 Class Reference	42
5.36.1 Member Function Documentation	42
5.36.1.1 verifyLocalStates()	42
5.37 TestParser Class Reference	43
5.37.1 Member Function Documentation	43
5.37.1.1 parse()	43
5.38 TransitionTemplate Class Reference	43
5.39 Var Struct Reference	44
5.39.1 Detailed Description	44
5.40 VarAssignment Struct Reference	44
5.41 Verification Class Reference	45
5.41.1 Constructor & Destructor Documentation	46
5.41.1.1 Verification()	46
5.41.2 Member Function Documentation	46
5.41.2.1 addHistoryContext()	46
5.41.2.2 addHistoryDecision()	47
5.41.2.3 addHistoryMarkDecisionAsInvalid()	47
5.41.2.4 addHistoryStateStatus()	47
5.41.2.5 areGlobalStatesInTheSameEpistemicClass()	48
5.41.2.6 getEpistemicClassForGlobalState()	48
5.41.2.7 isAgentInCoalition()	49
5.41.2.8 isGlobalTransitionControlledByCoalition()	49
5.41.2.9 newHistoryMarkDecisionAsInvalid()	49
5.41.2.10 printCurrentHistory()	50
5.41.2.11 revertLastDecision()	50
5.41.2.12 undoHistoryUntil()	50

5.41.2.13 undoLastHistoryEntry()	51
5.41.2.14 verify()	51
5.41.2.15 verifyGlobalState()	51
5.41.2.16 verifyLocalStates()	52
<b>6 File Documentation</b>	<b>53</b>
6.1 Constants.hpp	53
6.2 GlobalModelGenerator.cpp File Reference	53
6.2.1 Detailed Description	53
6.3 GlobalModelGenerator.hpp File Reference	53
6.3.1 Detailed Description	54
6.4 GlobalModelGenerator.hpp	54
6.5 expressions.cc File Reference	54
6.5.1 Detailed Description	54
6.6 expressions.hpp File Reference	55
6.6.1 Detailed Description	56
6.7 expressions.hpp	56
6.8 nodes.cc File Reference	58
6.8.1 Detailed Description	59
6.9 nodes.hpp File Reference	59
6.9.1 Detailed Description	59
6.10 nodes.hpp	59
6.11 SeleneFormula.hpp	60
6.12 TestParser.hpp	61
6.13 Types.cc File Reference	61
6.13.1 Detailed Description	61
6.14 Types.hpp File Reference	62
6.14.1 Detailed Description	62
6.14.2 Enumeration Type Documentation	63
6.14.2.1 ConditionOperator	63
6.14.2.2 GlobalStateVerificationStatus	63
6.14.2.3 VarAssignmentType	63
6.15 Types.hpp	63
6.16 Utils.cpp File Reference	66
6.16.1 Detailed Description	67
6.16.2 Function Documentation	67
6.16.2.1 agentToString()	67
6.16.2.2 envToString()	67
6.16.2.3 localModelsToString()	67
6.16.2.4 outputGlobalModel()	68
6.17 Utils.hpp File Reference	68
6.17.1 Function Documentation	69

---

6.17.1.1 agentToString()	69
6.17.1.2 envToString()	69
6.17.1.3 localModelsToString()	69
6.17.1.4 outputGlobalModel()	70
6.18 Utils.hpp	70
6.19 Verification.cpp File Reference	70
6.19.1 Detailed Description	71
6.19.2 Function Documentation	71
6.19.2.1 dbgHistEnt()	71
6.19.2.2 dbgVerifStatus()	71
6.19.2.3 verStatusToStr()	72
6.20 Verification.hpp File Reference	72
6.20.1 Enumeration Type Documentation	73
6.20.1.1 HistoryEntryType	73
6.20.1.2 Mode	73
6.20.2 Function Documentation	73
6.20.2.1 verStatusToStr()	73
6.21 Verification.hpp	74
<b>Index</b>	<b>77</b>



# Chapter 1

## README

### To run:

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

### Configuration file:

```
build/config.txt
```



## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

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

Agent	9
AgentTemplate	10
Assignment	14
Cfg	15
Condition	15
EpistemicClass	15
ExprNode	25
ExprAdd	16
ExprAnd	17
ExprConst	17
ExprDiv	18
ExprEq	19
ExprGe	20
ExprGt	20
ExprIdent	21
ExprLe	22
ExprLt	23
ExprMul	23
ExprNe	24
ExprNot	25
ExprOr	26
ExprRem	27
ExprSub	28
Formula	28
GlobalModel	29
GlobalModelGenerator	29
GlobalState	34
GlobalTransition	35
HistoryDbg	36
HistoryEntry	38
LocalModels	39
LocalState	39
LocalStateTemplate	40
LocalTransition	41
SeleneFormula	42

SeleneFormula1 . . . . .	42
TestParser . . . . .	43
TransitionTemplate . . . . .	43
Var . . . . .	44
VarAssignment . . . . .	44
Verification . . . . .	45

## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">Agent</a>	Contains all data for a single <a href="#">Agent</a> , including id, name and all of the agents' variables . . . . .	9
<a href="#">AgentTemplate</a>	. . . . .	10
<a href="#">Assignment</a>	Represents an assingment . . . . .	14
<a href="#">Cfg</a>	. . . . .	15
<a href="#">Condition</a>	Represents a condition for <a href="#">LocalTransition</a> . . . . .	15
<a href="#">EpistemicClass</a>	Represents a single epistemic class . . . . .	15
<a href="#">ExprAdd</a>	Node for addition . . . . .	16
<a href="#">ExprAnd</a>	Node for AND operator . . . . .	17
<a href="#">ExprConst</a>	Node for a constant . . . . .	17
<a href="#">ExprDiv</a>	Node for division . . . . .	18
<a href="#">ExprEq</a>	Node for "==" operator . . . . .	19
<a href="#">ExprGe</a>	Node for ">=" operator . . . . .	20
<a href="#">ExprGt</a>	Node for ">" operator . . . . .	20
<a href="#">ExprIdent</a>	Node for an identifier . . . . .	21
<a href="#">ExprLe</a>	Node for "<=" operator . . . . .	22
<a href="#">ExprLt</a>	Node for "<" operator . . . . .	23
<a href="#">ExprMul</a>	Node for multiplication . . . . .	23
<a href="#">ExprNe</a>	Node for "!=" operator . . . . .	24
<a href="#">ExprNode</a>	Base node for expressions . . . . .	25

<a href="#">ExprNot</a>		
	Node for NOT operator . . . . .	25
<a href="#">ExprOr</a>		
	Node for OR operator . . . . .	26
<a href="#">ExprRem</a>		
	Node for modulo . . . . .	27
<a href="#">ExprSub</a>		
	Node for subtraction . . . . .	28
<a href="#">Formula</a>		
	Contains a coalition of <a href="#">Agent</a> from the formula . . . . .	28
<a href="#">GlobalModel</a>		
	Represents a global model, containing agents and a formula . . . . .	29
<a href="#">GlobalModelGenerator</a>		29
<a href="#">GlobalState</a>		
	Represents a single global state . . . . .	34
<a href="#">GlobalTransition</a>		
	Represents a single global transition . . . . .	35
<a href="#">HistoryDbg</a>		36
<a href="#">HistoryEntry</a>		
	Structure used to save model traversal history . . . . .	38
<a href="#">LocalModels</a>		
	Represents a single local model, contains all agents and variables . . . . .	39
<a href="#">LocalState</a>		
	Represents a single <a href="#">LocalState</a> , containing id, name and internal variables . . . . .	39
<a href="#">LocalStateTemplate</a>		40
<a href="#">LocalTransition</a>		
	Represents a single local transition, containing id, global name, local name, is shared and count of the appearances . . . . .	41
<a href="#">SeleneFormula</a>		42
<a href="#">SeleneFormula1</a>		42
<a href="#">TestParser</a>		43
<a href="#">TransitionTemplate</a>		43
<a href="#">Var</a>		
	Represents a variable in the model, containing name, initial value and persistence . . . . .	44
<a href="#">VarAssignment</a>		44
<a href="#">Verification</a>		45

## Chapter 4

# File Index

### 4.1 File List

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

<a href="#">Constants.hpp</a>	53
<a href="#">GlobalModelGenerator.cpp</a>	
Generator of a global model. Class for initializing and generating a global model	53
<a href="#">GlobalModelGenerator.hpp</a>	
Generator of a global model. Class for initializing and generating a global model	53
<a href="#">expressions.cc</a>	
Eval and helper class for expressions. Eval and helper class for expressions	54
<a href="#">expressions.hpp</a>	
Eval and helper class for expressions. Eval and helper class for expressions	55
<a href="#">nodes.cc</a>	
Parser templates. Class for setting up a new objects from a parser	58
<a href="#">nodes.hpp</a>	
Parser templates. Class for setting up a new objects from a parser	59
<a href="#">SeleneFormula.hpp</a>	60
<a href="#">TestParser.hpp</a>	61
<a href="#">Types.cc</a>	
Custom data structures. Data structures and classes containing model data	61
<a href="#">Types.hpp</a>	
Custom data structures. Data structures and classes containing model data	62
<a href="#">Utils.cpp</a>	
Utility functions. A collection of utility functions to use in the project	66
<a href="#">Utils.hpp</a>	68
<a href="#">Verification.cpp</a>	
Class for verification of the formula on a model. Class for verification of the specified formula on a specified model	70
<a href="#">Verification.hpp</a>	72





## 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 <a href="#">LocalState</a> 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

### Public Member Functions

- [AgentTemplate](#) ()

Constructor for an *AgentTemplate*.

- virtual *AgentTemplate* & *setId*(string \_id)  
*Sets the identifier of an agent.*
- virtual *AgentTemplate* & *setInitState*(string \_startState)  
*Sets initial state of an agent.*
- virtual *AgentTemplate* & *addLocal*(set< string > \*variables)  
*Adds local variables to the agent.*
- virtual *AgentTemplate* & *addPersistent*(set< string > \*variables)  
*Adds persistent variables to the agent.*
- virtual *AgentTemplate* & *addInitial*(set< *Assignment* \* > \*assigns)  
???
- virtual *AgentTemplate* & *addTransition*(*TransitionTemplate* \*\_transition)  
???
- virtual *Agent* \* *generateAgent*(int id)  
*Generates an agent for the model.*

## 5.2.1 Member Function Documentation

### 5.2.1.1 addInitial()

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

???

#### Parameters

<i>assigns</i>	???
----------------	-----

#### Returns

???

### 5.2.1.2 addLocal()

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

Adds local variables to the agent.

#### Parameters

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

**Returns**

Returns itself.

**5.2.1.3 addPersistent()**

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

Adds persistent variables to the agent.

**Parameters**

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

**Returns**

Returns itself.

**5.2.1.4 addTransition()**

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

???

**Parameters**

<i>_transition</i>	???
--------------------	-----

**Returns**

???

**5.2.1.5 generateAgent()**

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

Generates an agent for the model.

## Parameters

<i>id</i>	Identifier of the new <a href="#">Agent</a> .
-----------	---

## Returns

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

**5.2.1.6 setIdent()**

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

Sets the identifier of an agent.

## Parameters

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

## Returns

Returns itself.

**5.2.1.7 setInitState()**

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

Sets initial state of an agent.

## Parameters

<i>_initState</i>	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)

### 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.

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**
- 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 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**  
*???*

### 5.6.1 Detailed Description

Represents a single epistemic class.

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

- [Types.hpp](#)

## 5.7 ExprAdd Class Reference

Node for addition.

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

### Public Member Functions

- **ExprAdd** ([ExprNode](#) \* \_larg, [ExprNode](#) \* \_rarg)
- virtual int **eval** ([Environment](#) &env)
- virtual int **eval** ([Environment](#) &env)=0

### 5.7.1 Detailed Description

Node for addition.

### 5.7.2 Member Function Documentation

#### 5.7.2.1 eval()

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

Implements [ExprNode](#).

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

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



## 5.8 ExprAnd Class Reference

Node for AND operator.

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

### Public Member Functions

- **ExprAnd** ([ExprNode](#) \*\_larg, [ExprNode](#) \*\_rarg)
- virtual int **eval** ([Environment](#) &env)
- virtual int **eval** ([Environment](#) &env)=0

### 5.8.1 Detailed Description

Node for AND operator.

### 5.8.2 Member Function Documentation

#### 5.8.2.1 eval()

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

Implements [ExprNode](#).

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

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

## 5.9 ExprConst Class Reference

Node for a constant.

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

### Public Member Functions

- **ExprConst** (int \_val)
- virtual int **eval** ([Environment](#) &env)
- virtual int **eval** ([Environment](#) &env)=0

### 5.9.1 Detailed Description

Node for a constant.

### 5.9.2 Member Function Documentation

#### 5.9.2.1 eval()

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

Implements [ExprNode](#).

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

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

## 5.10 ExprDiv Class Reference

Node for division.

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

### Public Member Functions

- **ExprDiv** ([ExprNode](#) \*\_larg, [ExprNode](#) \*\_rarg)
- virtual int **eval** ([Environment](#) &env)
- virtual int **eval** ([Environment](#) &env)=0

#### 5.10.1 Detailed Description

Node for division.

#### 5.10.2 Member Function Documentation

### 5.10.2.1 eval()

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

Implements [ExprNode](#).

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

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

## 5.11 ExprEq Class Reference

Node for "==" operator.

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

### Public Member Functions

- **ExprEq** ([ExprNode](#) \*\_larg, [ExprNode](#) \*\_rarg)
- virtual int **eval** ([Environment](#) &env)
- virtual int **eval** ([Environment](#) &env)=0

### 5.11.1 Detailed Description

Node for "==" operator.

### 5.11.2 Member Function Documentation

#### 5.11.2.1 eval()

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

Implements [ExprNode](#).

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

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

## 5.12 ExprGe Class Reference

Node for ">=" operator.

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

### Public Member Functions

- **ExprGe** ([ExprNode](#) \*\_larg, [ExprNode](#) \*\_rarg)
- virtual int **eval** ([Environment](#) &env)
- virtual int **eval** ([Environment](#) &env)=0

### 5.12.1 Detailed Description

Node for ">=" operator.

### 5.12.2 Member Function Documentation

#### 5.12.2.1 eval()

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

Implements [ExprNode](#).

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

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

## 5.13 ExprGt Class Reference

Node for ">" operator.

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

### Public Member Functions

- **ExprGt** ([ExprNode](#) \*\_larg, [ExprNode](#) \*\_rarg)
- virtual int **eval** ([Environment](#) &env)
- virtual int **eval** ([Environment](#) &env)=0

### 5.13.1 Detailed Description

Node for ">" operator.

### 5.13.2 Member Function Documentation

#### 5.13.2.1 eval()

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

Implements [ExprNode](#).

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

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

## 5.14 ExprIdent Class Reference

Node for an identifier.

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

### Public Member Functions

- **ExprIdent** (string \_ident)
- virtual int **eval** ([Environment](#) &env)
- virtual int **eval** ([Environment](#) &env)=0

### 5.14.1 Detailed Description

Node for an identifier.

### 5.14.2 Member Function Documentation

#### 5.14.2.1 eval()

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

Implements [ExprNode](#).

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

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

## 5.15 ExprLe Class Reference

Node for "<=" operator.

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

### Public Member Functions

- **ExprLe** ([ExprNode](#) \* \_larg, [ExprNode](#) \* \_rarg)
- virtual int **eval** ([Environment](#) &env)
- virtual int **eval** ([Environment](#) &env)=0

#### 5.15.1 Detailed Description

Node for "<=" operator.

#### 5.15.2 Member Function Documentation

##### 5.15.2.1 eval()

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

Implements [ExprNode](#).

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

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

## 5.16 ExprLt Class Reference

Node for "<" operator.

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

### Public Member Functions

- **ExprLt** ([ExprNode](#) \*\_larg, [ExprNode](#) \*\_rarg)
- virtual int **eval** ([Environment](#) &env)
- virtual int **eval** ([Environment](#) &env)=0

#### 5.16.1 Detailed Description

Node for "<" operator.

#### 5.16.2 Member Function Documentation

##### 5.16.2.1 eval()

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

Implements [ExprNode](#).

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

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

## 5.17 ExprMul Class Reference

Node for multiplication.

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

### Public Member Functions

- **ExprMul** ([ExprNode](#) \*\_larg, [ExprNode](#) \*\_rarg)
- virtual int **eval** ([Environment](#) &env)
- virtual int **eval** ([Environment](#) &env)=0

### 5.17.1 Detailed Description

Node for multiplication.

### 5.17.2 Member Function Documentation

#### 5.17.2.1 eval()

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

Implements [ExprNode](#).

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

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

## 5.18 ExprNe Class Reference

Node for "!=" operator.

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

### Public Member Functions

- **ExprNe** ([ExprNode](#) \* \_larg, [ExprNode](#) \* \_rarg)
- virtual int **eval** ([Environment](#) &env)
- virtual int **eval** ([Environment](#) &env)=0

### 5.18.1 Detailed Description

Node for "!=" operator.

### 5.18.2 Member Function Documentation



### 5.18.2.1 eval()

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

Implements [ExprNode](#).

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

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

## 5.19 ExprNode Class Reference

Base node for expressions.

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

### Public Member Functions

- virtual int **eval** ([Environment](#) &env)=0

### 5.19.1 Detailed Description

Base node for expressions.

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

- [expressions.hpp](#)

## 5.20 ExprNot Class Reference

Node for NOT operator.

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

### Public Member Functions

- **ExprNot** ([ExprNode](#) \*\_arg)
- virtual int **eval** ([Environment](#) &env)
- virtual int **eval** ([Environment](#) &env)=0

### 5.20.1 Detailed Description

Node for NOT operator.

### 5.20.2 Member Function Documentation

#### 5.20.2.1 eval()

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

Implements [ExprNode](#).

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

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

## 5.21 ExprOr Class Reference

Node for OR operator.

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

### Public Member Functions

- **ExprOr** ([ExprNode](#) \*\_larg, [ExprNode](#) \*\_rarg)
- virtual int **eval** ([Environment](#) &env)
- virtual int **eval** ([Environment](#) &env)=0

#### 5.21.1 Detailed Description

Node for OR operator.

#### 5.21.2 Member Function Documentation

### 5.21.2.1 eval()

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

Implements [ExprNode](#).

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

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

## 5.22 ExprRem Class Reference

Node for modulo.

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

### Public Member Functions

- **ExprRem** ([ExprNode](#) \*\_larg, [ExprNode](#) \*\_rarg)
- virtual int **eval** ([Environment](#) &env)
- virtual int **eval** ([Environment](#) &env)=0

### 5.22.1 Detailed Description

Node for modulo.

### 5.22.2 Member Function Documentation

#### 5.22.2.1 eval()

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

Implements [ExprNode](#).

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

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

## 5.23 ExprSub Class Reference

Node for subtraction.

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

### Public Member Functions

- **ExprSub** ([ExprNode](#) \*\_larg, [ExprNode](#) \*\_rarg)
- virtual int **eval** ([Environment](#) &env)
- virtual int **eval** ([Environment](#) &env)=0

### 5.23.1 Detailed Description

Node for subtraction.

### 5.23.2 Member Function Documentation

#### 5.23.2.1 eval()

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

Implements [ExprNode](#).

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

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

## 5.24 Formula Struct Reference

Contains a coalition of [Agent](#) from the formula.

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

### Public Attributes

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

### 5.24.1 Detailed Description

Contains a coalition of [Agent](#) from the formula.

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

- [Types.hpp](#)

## 5.25 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](#).*

### 5.25.1 Detailed Description

Represents a global model, containing agents and a formula.

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

- [Types.hpp](#)

## 5.26 GlobalModelGenerator Class Reference

### 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)`  
*???*
- `void expandAllStates ()`  
*???*
- `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)`  
*???*
- `void generateGlobalTransitions (GlobalState *fromGlobalState, set< LocalTransition * > localTransitions, map< Agent *, vector< LocalTransition * > > transitionsByAgent)`  
*???*
- `bool checkLocalTransitionConditions (LocalTransition *localTransition, GlobalState *globalState)`  
*???*
- `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)`  
*???*

## Protected Attributes

- `LocalModels * localModels`  
*`LocalModels` used in `initModel`.*
- `Formula * formula`  
*`Formula` used in `initModel`.*
- `GlobalModel * globalModel`  
*`GlobalModel` created in `initModel`.*

## 5.26.1 Member Function Documentation

### 5.26.1.1 checkLocalTransitionConditions()

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

???

#### Parameters

<code>localTransition</code>	???
<code>globalState</code>	???

**Returns**

???

**5.26.1.2 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 <a href="#">LocalState</a> and pointer to and <a href="#">Agent</a> to turn into a hash.
--------------------	--

**Returns**

Returns a string with a hash.

**5.26.1.3 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 <a href="#">LocalState</a> to turn into a hash.
--------------------	---

**Returns**

Returns a string with a hash.

**5.26.1.4 expandState()**

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

???

## Parameters

<i>state</i>	???
--------------	-----

**5.26.1.5 findGlobalStateInEpistemicClass()**

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

???

## Parameters

<i>localStates</i>	???
<i>epistemicClass</i>	???

## Returns

???

**5.26.1.6 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>	<a href="#">Agent</a> for which to check the existence of an epistemic class.

## Returns

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

**5.26.1.7 generateGlobalTransitions()**

```
void GlobalModelGenerator::generateGlobalTransitions (
    GlobalState * fromGlobalState,
```



```
set< LocalTransition * > localTransitions,
map< Agent *, vector< LocalTransition * > > transitionsByAgent ) [protected]
```

???

#### Parameters

<i>fromGlobalState</i>	???
<i>localTransitions</i>	???
<i>transitionsByAgent</i>	???

#### 5.26.1.8 generateInitState()

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

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

#### Returns

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

#### 5.26.1.9 generateStateFromLocalStates()

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

???

#### Parameters

<i>localStates</i>	???
<i>viaLocalTransitions</i>	???
<i>prevGlobalState</i>	???

#### Returns

???

#### 5.26.1.10 getCurrentGlobalModel()

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

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

**Returns**

Returns a pointer to a global model.

**5.26.1.11 getFormula()**

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

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

**Returns****5.26.1.12 initModel()**

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

Initializes a global model from local models and a formula.

**Parameters**

<i>localModels</i>	Pointer to <a href="#">LocalModels</a> that will construct a global model.
<i>formula</i>	Pointer to a <a href="#">Formula</a> 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](#)

**5.27 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**  
*???*
- **[GlobalStateVerificationStatus](#) verificationStatus**  
*Current verification status of this state.*
- **set< [GlobalTransition](#) \* > globalTransitions**  
*Every [GlobalTransition](#) in the model.*
- **set< [LocalState](#) \* > localStates**  
*???*

### 5.27.1 Detailed Description

Represents a single global state.

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

- [Types.hpp](#)

## 5.28 GlobalTransition Struct Reference

Represents a single global transition.

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

## Public Attributes

- **int id**  
*Identifier of the transition.*
- **bool isInvalidDecision**  
*???*
- **[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**  
*???*

### 5.28.1 Detailed Description

Represents a single global transition.

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

- [Types.hpp](#)

## 5.29 HistoryDbg Class Reference

### 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.*

### 5.29.1 Member Function Documentation

#### 5.29.1.1 addEntry()

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

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

#### Parameters

<i>entry</i>	A pointer to the <a href="#">HistoryEntry</a> that will be added to the history.
--------------	--

### 5.29.1.2 cloneEntry()

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

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

#### Parameters

<i>entry</i>	A pointer to a <a href="#">HistoryEntry</a> to be checked.
--------------	--

#### Returns

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

### 5.29.1.3 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 <a href="#">HistoryEntry</a> that is supposed to be marked.
<i>chr</i>	A char that will be made into a pair with a <a href="#">HistoryEntry</a> .

### 5.29.1.4 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](#)

## 5.30 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](#).*

### 5.30.1 Detailed Description

Structure used to save model traversal history.

### 5.30.2 Member Function Documentation

### 5.30.2.1 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](#)

## 5.31 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.*
- map< string, [Var](#) \* > **vars**  
*A map of variable names to [Var](#).*

### 5.31.1 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](#)

## 5.32 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< Var \*, int > vars**  
*???*
- **map< string, int > environment**  
*???*
- **Agent \* agent**  
*Binding to an [Agent](#).*
- **set< LocalTransition \* > localTransitions**  
*Binding to the set of [LocalTransition](#).*

### 5.32.1 Detailed Description

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

### 5.32.2 Member Function Documentation

#### 5.32.2.1 compare()

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

Function comparing two states.

#### Parameters

<i>state</i>	A pointer to <a href="#">LocalState</a> 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](#)

## 5.33 LocalStateTemplate Class Reference

### Public Attributes

- **string name**



- set< [TransitionTemplate](#) \* > **transitions**

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

- [nodes.hpp](#)

## 5.34 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**  
???
- set< [VarAssignment](#) \* > **varAssignments**  
???
- [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**  
???

### 5.34.1 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](#)

## 5.35 SeleneFormula Class Reference

### Public Member Functions

- virtual bool **verifyLocalStates** (set< [LocalState](#) \* > \*localStates)=0
- [LocalState](#) \* **getLocalStateForAgent** (string agentName, set< [LocalState](#) \* > \*localStates)
- int **getLocalStateVar** (string varName, [LocalState](#) \*localState)
- bool **implication** (bool left, bool right)

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

- SeleneFormula.hpp
- SeleneFormula.cpp

## 5.36 SeleneFormula1 Class Reference

### Public Member Functions

- bool [verifyLocalStates](#) (set< [LocalState](#) \* > \*localStates)

### Public Member Functions inherited from [SeleneFormula](#)

- virtual bool **verifyLocalStates** (set< [LocalState](#) \* > \*localStates)=0
- [LocalState](#) \* **getLocalStateForAgent** (string agentName, set< [LocalState](#) \* > \*localStates)
- int **getLocalStateVar** (string varName, [LocalState](#) \*localState)
- bool **implication** (bool left, bool right)

### 5.36.1 Member Function Documentation

#### 5.36.1.1 [verifyLocalStates\(\)](#)

```
bool SeleneFormula1::verifyLocalStates (
    set< LocalState * > * localStates ) [virtual]
```

Implements [SeleneFormula](#).

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

- SeleneFormula.hpp
- SeleneFormula.cpp

## 5.37 TestParser Class Reference

### Public Member Functions

- **TestParser** ()  
*TestParser constructor.*
- **~TestParser** ()  
*TestParser destructor.*
- **LocalModels** \* **parse** (string fileName)  
*Parses a file with given name into a usable model.*

### 5.37.1 Member Function Documentation

#### 5.37.1.1 parse()

```
LocalModels * TestParser::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:

- TestParser.hpp
- TestParser.cc

## 5.38 TransitionTemplate Class Reference

### Public Member Functions

- **TransitionTemplate** (int \_shared, string \_patternName, string \_matchName, string \_startState, string \_↔  
endState, ExprNode \* \_cond, set< Assignment \* > \* \_assign)

## Public Attributes

- int **shared**
- string **patternName**
- string **matchName**
- string **startState**
- string **endState**
- [ExprNode](#) \* **condition**
- set< [Assignment](#) \* > \* **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 VarAssignment Struct Reference

## Public Attributes

- [Var](#) \* **dstVar**
- [VarAssignmentType](#) **type**
- [Var](#) \* **srcVar**
- int **value**

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

- [Types.hpp](#)

## 5.41 Verification Class Reference

### 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.*

## 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.*
- [SeleneFormula](#) \* **seleneFormula**  
*Temporary solve for data input.*
- [HistoryEntry](#) \* **historyStart**  
*Pointer to the start of model traversal history.*
- [HistoryEntry](#) \* **historyEnd**  
*Pointer to the end of model traversal history.*

## 5.41.1 Constructor & Destructor Documentation

### 5.41.1.1 Verification()

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

Constructor for [Verification](#).

Parameters

<i>generator</i>	Pointer to <a href="#">GlobalModelGenerator</a>
------------------	---

## 5.41.2 Member Function Documentation

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

**5.41.2.2 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 <a href="#">GlobalState</a> of the model.
<i>decision</i>	Pointer to a <a href="#">GlobalTransition</a> that is to be recorded in the decision history.

**5.41.2.3 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 <a href="#">GlobalState</a> of the model.
<i>decision</i>	Pointer to a transition <a href="#">GlobalTransition</a> selected by the algorithm.

**5.41.2.4 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 <a href="#">GlobalState</a> of the model.
<i>prevStatus</i>	Previous GlobalStateVerificationStatus to be logged.
<i>newStatus</i>	New GlobalStateVerificationStatus to be logged.

**5.41.2.5 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 <a href="#">GlobalState</a> .
<i>globalState2</i>	Pointer to the second <a href="#">GlobalState</a> .

## 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](#).

**5.41.2.6 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 <a href="#">GlobalState</a> 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.



**5.41.2.7 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 <a href="#">Agent</a> that is to be checked.
--------------	--

**Returns**

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

**5.41.2.8 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 <a href="#">GlobalTransition</a> in a model.
-------------------------	---

**Returns**

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

**5.41.2.9 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 <a href="#">GlobalState</a> of the model.
<i>decision</i>	Pointer to a transition <a href="#">GlobalTransition</a> selected by the algorithm.

**Returns**

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

**5.41.2.10 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.
--------------	--

**5.41.2.11 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.

**5.41.2.12 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 <a href="#">HistoryEntry</a> 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.

### 5.41.2.13 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.
-------------------	--

### 5.41.2.14 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.

### 5.41.2.15 verifyGlobalState()

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

Recursively verifies [GlobalState](#).

#### Parameters

<i>globalState</i>	Pointer to a <a href="#">GlobalState</a> 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.

### 5.41.2.16 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 <a href="#">LocalState</a> .
--------------------	--

#### Returns

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

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

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

## Chapter 6

# File Documentation

### 6.1 Constants.hpp

```
00001 #ifndef SELENE_CONSTANTS
00002 #define SELENE_CONSTANTS
00003
00004 #define VERBOSE 0
00005 // #define OUTPUT_LOCAL_MODELS 1
00006 // #define OUTPUT_GLOBAL_MODEL 0 // warning: it will call expandAllStates()
00007 // #define MODE 2 // 1 = only generate; 2 = verify
00008
00009 // Model id
00010 // 1 = src/examples/trains/Trains.txt
00011 // 2 = src/examples/ssvr/Selene_Select_Vote_Revoting_lv_lcv_3c_3rev_share.txt
00012 // 3 = src/examples/svote/Simple_voting.txt
00013 // #define MODEL_ID 1
00014
00015 struct Cfg{
00016     char* fname;
00017     char stv_mode;
00018     bool output_local_models;
00019     bool output_global_model;
00020     int model_id; // <-- this is temporary member (used in Verification.cpp for a hardcoded formula)
00021 };
00022
00023 #endif // SELENE_CONSTANTS
```

### 6.2 GlobalModelGenerator.cpp File Reference

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

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

#### 6.2.1 Detailed Description

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

### 6.3 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](#)

### 6.3.1 Detailed Description

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

## 6.4 GlobalModelGenerator.hpp

[Go to the documentation of this file.](#)

```

00001
00007 #ifndef SELENE_GLOBAL_MODEL_GENERATOR
00008 #define SELENE_GLOBAL_MODEL_GENERATOR
00009
00010 #include "Constants.hpp"
00011 #include "Types.hpp"
00012
00013 using namespace std;
00014
00015 class GlobalModelGenerator {
00016 public:
00017     GlobalModelGenerator();
00018     ~GlobalModelGenerator();
00019     GlobalState* initModel(LocalModels* localModels, Formula* formula);
00020     void expandState(GlobalState* state);
00021     void expandAllStates();
00022     GlobalModel* getCurrentGlobalModel();
00023     Formula* getFormula();
00024
00025 protected:
00026     LocalModels* localModels;
00027     Formula* formula;
00028     GlobalModel* globalModel;
00029     GlobalState* generateInitState();
00030     GlobalState* generateStateFromLocalStates(set<LocalState*>* localStates, set<LocalTransition*>*
00031 viaLocalTransitions, GlobalState* prevGlobalState);
00032     void generateGlobalTransitions(GlobalState* fromGlobalState, set<LocalTransition*>
00033 localTransitions, map<Agent*, vector<LocalTransition*>> transitionsByAgent);
00034     bool checkLocalTransitionConditions(LocalTransition* localTransition, GlobalState* globalState);
00035     string computeEpistemicClassHash(set<LocalState*>* localStates, Agent* agent);
00036     string computeGlobalStateHash(set<LocalState*>* localStates);
00037     EpistemicClass* findOrCreateEpistemicClass(set<LocalState*>* localStates, Agent* agent);
00038     GlobalState* findGlobalStateInEpistemicClass(set<LocalState*>* localStates, EpistemicClass*
00039 epistemicClass);
00040 };
00041
00042 #endif // SELENE_GLOBAL_MODEL_GENERATOR

```

## 6.5 expressions.cc File Reference

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

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

### 6.5.1 Detailed Description

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

## 6.6 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**  
*???*

## 6.6.1 Detailed Description

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

## 6.7 expressions.hpp

[Go to the documentation of this file.](#)

```

00001
00007 #ifndef __EXPRESSIONS_H
00008 #define __EXPRESSIONS_H
00009
00010 #include <string>
00011 #include <map>
00012
00013 using namespace std;
00014
00016 typedef map<string, int> Environment;
00017
00018 // węzeł bazowy dla wyrażeń
00019
00021 class ExprNode {
00022
00023     public:
00024         // metoda do wyliczenia wartości wyrażenia zależna od typu węzła
00025         virtual int eval( Environment& env ) = 0;
00026 };
00027
00028 // węzeł dla stałej
00029
00031 class ExprConst: public ExprNode {
00032
00033     // argumenty
00034
00036     int val;
00037
00038     public:
00039         ExprConst(int _val): val(_val) {};
00040         virtual int eval( Environment& env );
00041 };
00042
00043 // węzeł dla identyfikatora
00044
00046 class ExprIdent: public ExprNode {
00047
00048     // argumenty
00049
00051     string ident;
00052
00053     public:
00054         ExprIdent(string _ident): ident(_ident) {};
00055         virtual int eval( Environment& env );
00056 };
00057
00058 // węzeł dla dodawań
00059
00061 class ExprAdd: public ExprNode {
00062
00063     // argumenty
00064
00066     ExprNode *larg, *rarg;
00067
00068     public:
00069         ExprAdd(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00070         virtual int eval( Environment& env );
00071 };
00072
00073 // węzeł dla odejmowań
00074
00076 class ExprSub: public ExprNode {
00077
00078     // argumenty
00079
00081     ExprNode *larg, *rarg;
00082
00083     public:
00084         ExprSub(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00085         virtual int eval( Environment& env );
00086 };
00087

```



```

00088 // węzeł dla mnożeń
00089
00091 class ExprMul: public ExprNode {
00092
00093     // argumenty
00094
00096     ExprNode *larg, *rarg;
00097
00098     public:
00099         ExprMul(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00100         virtual int eval( Environment& env );
00101 };
00102
00103 // węzeł dla dzielenia
00104
00106 class ExprDiv: public ExprNode {
00107
00108     // argumenty
00109
00111     ExprNode *larg, *rarg;
00112
00113     public:
00114         ExprDiv(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00115         virtual int eval( Environment& env );
00116 };
00117
00118 // węzeł dla reszty z dzielenia
00119
00121 class ExprRem: public ExprNode {
00122
00123     // argumenty
00124
00126     ExprNode *larg, *rarg;
00127
00128     public:
00129         ExprRem(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00130         virtual int eval( Environment& env );
00131 };
00132
00133 // węzeł dla operatora AND
00134
00136 class ExprAnd: public ExprNode {
00137
00138     // argumenty
00139
00141     ExprNode *larg, *rarg;
00142
00143     public:
00144         ExprAnd(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00145         virtual int eval( Environment& env );
00146 };
00147
00148 // węzeł dla operatora OR
00149
00151 class ExprOr: public ExprNode {
00152
00153     // argumenty
00154
00156     ExprNode *larg, *rarg;
00157
00158     public:
00159         ExprOr(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00160         virtual int eval( Environment& env );
00161 };
00162
00163 // węzeł dla operatora NOT
00164
00166 class ExprNot: public ExprNode {
00167
00168     // argumenty
00169
00171     ExprNode *arg;
00172
00173     public:
00174         ExprNot(ExprNode *_arg): arg(_arg) {};
00175         virtual int eval( Environment& env );
00176 };
00177
00178 // węzeł dla operatora "=="
00179
00181 class ExprEq: public ExprNode {
00182
00183     // argumenty
00184
00186     ExprNode *larg, *rarg;
00187
00188     public:

```

```

00189     ExprEq(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00190     virtual int eval( Environment& env );
00191 };
00192
00193 // węzeł dla operatora "!="
00194
00196 class ExprNe: public ExprNode {
00197
00198     // argumenty
00199
00201     ExprNode *larg, *rarg;
00202
00203     public:
00204         ExprNe(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00205         virtual int eval( Environment& env );
00206 };
00207
00208 // węzeł dla operatora "<"
00209
00211 class ExprLt: public ExprNode {
00212
00213     // argumenty
00214
00216     ExprNode *larg, *rarg;
00217
00218     public:
00219         ExprLt(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00220         virtual int eval( Environment& env );
00221 };
00222
00223 // węzeł dla operatora "<="
00224
00226 class ExprLe: public ExprNode {
00227
00228     // argumenty
00229
00231     ExprNode *larg, *rarg;
00232
00233     public:
00234         ExprLe(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00235         virtual int eval( Environment& env );
00236 };
00237
00238 // węzeł dla operatora ">"
00239
00241 class ExprGt: public ExprNode {
00242
00243     // argumenty
00244
00246     ExprNode *larg, *rarg;
00247
00248     public:
00249         ExprGt(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00250         virtual int eval( Environment& env );
00251 };
00252
00253 // węzeł dla operatora ">="
00254
00256 class ExprGe: public ExprNode {
00257
00258     // argumenty
00259
00261     ExprNode *larg, *rarg;
00262
00263     public:
00264         ExprGe(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00265         virtual int eval( Environment& env );
00266 };
00267
00268
00269 #endif

```

## 6.8 nodes.cc File Reference

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

```

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

```

### 6.8.1 Detailed Description

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

## 6.9 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](#)
- class [LocalStateTemplate](#)
- class [AgentTemplate](#)

### 6.9.1 Detailed Description

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

## 6.10 nodes.hpp

[Go to the documentation of this file.](#)

```
00001
00007 #ifndef __NODES_H
00008 #define __NODES_H
00009
00010 #include <string>
00011 #include <set>
00012 #include <map>
00013 #include "expressions.hpp"
00014
00015 #include "../Types.hpp"
00016
00017 using namespace std;
00018
00019 /* Klasa reprezentująca przypisanie */
00021 class Assignment {
00022     public:
00024         string ident;
00025         // do czego przypisujemy
00026
00028         ExprNode *value;
00029         // co przypisujemy
00030
00034         Assignment(string _ident, ExprNode *_exp): ident(_ident), value(_exp) {};
00035
00036         // wykonaj przypisanie w danym środowisku
00037         virtual void assign(Environment &env) {
00038             env[ident]=value->eval(env);
00039         };
00039
```

```

00040 };
00041
00042 /* Klasa reprezentująca meta-tranzycję */
00043 class TransitionTemplate {
00044 public:
00045     // jeśli -1 to nie ma dzielenia, w p.p. wartość określa łączną liczbę wymaganych agentów
00046     int shared;
00047     // nazwa wzorca
00048     string patternName;
00049     // nazwa do wyszukiwania dla shared
00050     string matchName;
00051     // nazwa stanu początkowego i końcowego
00052     string startState;
00053     string endState;
00054     // wyrażenie warunkowe
00055     ExprNode *condition;
00056     // lista przypisań wartości
00057     set<Assignment*> *assignments;
00058
00059     TransitionTemplate(int _shared, string _patternName, string _matchName, string _startState,
00060         string _endState, ExprNode *_cond, set<Assignment*> *_assign):
00061         shared(_shared), patternName(_patternName), matchName(_matchName),
00062         startState(_startState), endState(_endState), condition(_cond), assignments(_assign) {};
00063 };
00064
00065 class LocalStateTemplate {
00066 public:
00067     string name;
00068     set<TransitionTemplate*> transitions;
00069 };
00070
00071 /* Klasa reprezentująca pojedynczego agenta po wczytaniu jego opisu z pliku */
00072 class AgentTemplate {
00073     // identyfikator agenta
00074     string ident;
00075     // stan startowy
00076     string initState;
00077     // zbiór zmiennych lokalnych (local)
00078     set<string*> localVars;
00079     // zbiór zmiennych trwałych (persistent)
00080     set<string*> persistentVars;
00081     // początkowa inicjacja
00082     set<Assignment*> *initialAssignments;
00083     // zbiór tranzycji
00084     set<TransitionTemplate*> *transitions;
00085
00086     // mapa stanów lokalnych potrzebna do wygenerowania modelu
00087     map<string, LocalStateTemplate*> localStateTemplates;
00088
00089     // metoda wyznaczająca węzeł kolejny do danego, zależnie od tranzycji
00090     virtual LocalState* genNextState(LocalState *state, TransitionTemplate *trans);
00091
00092 public:
00093     AgentTemplate();
00094
00095     // ustaw identyfikator agenta
00096     virtual AgentTemplate& setIdent(string _ident);
00097     // ustaw identyfikator agenta
00098     virtual AgentTemplate& setInitState(string _startState);
00099     // dodaj zmienną/zmienne lokalne
00100     virtual AgentTemplate& addLocal(set<string*> *variables);
00101     // dodaj zmienne trwałe
00102     virtual AgentTemplate& addPersistent(set<string*> *variables);
00103     // dodaj początkowe inicjacje
00104     virtual AgentTemplate& addInitial(set<Assignment*> *assigns);
00105     // dodaj tranzycję
00106     virtual AgentTemplate& addTransition(TransitionTemplate *_transition);
00107
00108     // wygeneruj agenta do modelu
00109     virtual Agent* generateAgent(int id) ;
00110 };
00111
00112 #endif

```

## 6.11 SeleneFormula.hpp

```

00001 #ifndef SELENE_SELENE_FORMULA
00002 #define SELENE_SELENE_FORMULA
00003
00004 #include "Types.hpp"
00005
00006

```

```

00007
00008
00009
00010 class SeleneFormula {
00011 public:
00012     SeleneFormula();
00013     ~SeleneFormula();
00014     virtual bool verifyLocalStates(set<LocalState*>* localStates) = 0;
00015     LocalState* getLocalStateForAgent(string agentName, set<LocalState*>* localStates);
00016     int getLocalStateVar(string varName, LocalState* localState);
00017     inline bool implication(bool left, bool right);
00018 };
00019
00020
00021
00022
00023
00024 class SeleneFormula1 : public SeleneFormula {
00025 public:
00026     SeleneFormula1();
00027     ~SeleneFormula1();
00028     bool verifyLocalStates(set<LocalState*>* localStates);
00029 protected:
00030 };
00031
00032
00033
00034
00035
00036 #endif // SELENE_SELENE_FORMULA

```

## 6.12 TestParser.hpp

```

00001
00007 #ifndef __TESTPARSER_HPP
00008 #define __TESTPARSER_HPP
00009
00010 #include "Types.hpp"
00011
00012 using namespace std;
00013
00014 class TestParser {
00015 public:
00016     TestParser();
00017     ~TestParser();
00018     LocalModels* parse(string fileName);
00019
00020 protected:
00021     // @internal
00022 };
00023
00024 #endif

```

## 6.13 Types.cc File Reference

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

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

### 6.13.1 Detailed Description

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

## 6.14 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>
```

### Classes

- struct [Var](#)  
*Represents a variable in the model, containing name, initial value and persistence.*
- struct [Condition](#)  
*Represents a condition for [LocalTransition](#).*
- struct [Formula](#)  
*Contains a coalition of [Agent](#) from the 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 [VarAssignment](#)
- 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.*

#### 6.14.1 Detailed Description

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

## 6.14.2 Enumeration Type Documentation

### 6.14.2.1 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.

### 6.14.2.2 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.

### 6.14.2.3 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.

## 6.15 Types.hpp

[Go to the documentation of this file.](#)

```

00001
00007 #ifndef SELENE_TYPES
00008 #define SELENE_TYPES
00009
00010 #include <map>
00011 #include <set>
00012
00013 #include <stack>
00014 #include <string>
00015 #include <utility>
00016 #include <vector>
00017
00018 using namespace std;
00019
00020
00021 class Agent;
00022 class LocalState;
00023
00024 struct Condition;
00025 struct EpistemicClass;
00026 struct Formula;
00027 struct GlobalModel;
00028 struct GlobalState;
00029 struct GlobalTransition;
00030 struct LocalTransition;
00031 struct LocalModels;
00032 struct Var;
00033 struct VarAssignment;
00034
00036 enum ConditionOperator {
00037     Equals,
00038     NotEquals,
00039 };
00040
00042 enum GlobalStateVerificationStatus {
00043     UNVERIFIED,
00044     PENDING,
00045     VERIFIED_OK,
00046     VERIFIED_ERR,
00047 };
00048
00050 enum VarAssignmentType {
00051     FromVar,
00052     FromValue,
00053 };
00054
00056 struct Var {
00058     string name;
00059
00061     int initialValue;
00062
00064     bool persistent;
00065
00067     Agent *agent;
00068 };
00069
00071 struct Condition {
00073     Var* var;
00074
00076     ConditionOperator conditionOperator;
00077
00079     int comparedValue;
00080 };
00081
00083 struct Formula {
00085     set<Agent*> coalition;
00086 };
00087
00089 class Agent {
00090     public:
00092         int id;
00093
00095         string name;
00096
00098         set<Var*> vars;
00099
00103         Agent(int _id, string _name):id(_id), name(_name) {};
00104
00106         LocalState* initState;
00107
00109         vector<LocalState*> localStates; // localStates[i].id == i
00110
00112         vector<LocalTransition*> localTransitions; // localTransitions[i].id == i
00113
00114         // sprawdź, czy stan nie został już wygenerowany.
00115         LocalState* includesState(LocalState *state);
00116 };

```



```

00117
00119 class LocalState {
00120     public:
00121         // Data
00123         int id;
00124
00126         string name;
00127
00129         map<Var*, int> vars;
00130
00131         // alternatywna wersja - może wystarczy
00133         map<string, int> environment;
00134
00135         // komparator
00136         bool compare(LocalState *state);
00137
00138         // Bindings
00140         Agent* agent;
00141
00143         set<LocalTransition*> localTransitions;
00144 };
00145
00146 // to jest zbędne
00147 struct VarAssignment {
00148     Var* dstVar;
00149     VarAssignmentType type; // zbędne
00150     Var* srcVar;
00151     int value;
00152 };
00153
00155 struct LocalTransition {
00156     // Data
00158     int id;
00159
00161     string name;
00162
00164     string localName; // if empty => same as name
00165
00167     bool isShared;
00168
00170     int sharedCount;
00171
00173     set<Condition*> conditions;
00174
00176     set<VarAssignment*> varAsssignments;
00177
00178     // Bindings
00180     Agent* agent;
00181
00183     LocalState* from;
00184
00186     LocalState* to;
00187
00189     set<LocalTransition*> sharedLocalTransitions;
00190 };
00191
00193 struct LocalModels {
00195     vector<Agent*> agents;
00196     // agents[i].id == i
00197
00199     map<string, Var*> vars;
00200     // vars[str].name == str
00201 };
00202
00204 struct GlobalModel {
00205     // Data
00207     vector<Agent*> agents;
00208     // agents[i].id == i
00209
00211     Formula* formula;
00212
00213     // Bindings
00215     GlobalState* initState;
00216
00218     vector<GlobalState*> globalStates;
00219     // globalStates[i].id == i
00220
00222     vector<GlobalTransition*> globalTransitions;
00223     // globalTransitions[i].id == i
00224
00226     map<Agent*, map<string, EpistemicClass*>> epistemicClasses;
00227     // Agent* => (EpistemicClass->hash => EpistemicClass*)
00228 };
00229
00231 struct GlobalState {
00232     // Data
00234     int id;

```

```

00235
00237     string hash;
00238
00240     map<Var*, int> vars;
00241
00243     map<Agent*, EpistemicClass*> epistemicClasses;
00244
00246     bool isExpanded;
00247
00249     GlobalStateVerificationStatus verificationStatus;
00250
00251     // Bindings
00253     set<GlobalTransition*> globalTransitions;
00254
00256     set<LocalState*> localStates;
00257 };
00258
00260 struct GlobalTransition {
00261     // Data
00263     int id;
00264
00266     bool isInvalidDecision;
00267
00268     // Bindings
00270     GlobalState* from;
00271
00273     GlobalState* to;
00274
00276     set<LocalTransition*> localTransitions;
00277 };
00278
00280 struct EpistemicClass {
00282     string hash;
00283
00285     map<string, GlobalState*> globalStates;
00286     // GlobalState->hash => GlobalState*
00287
00289     GlobalTransition* fixedCoalitionTransition;
00290 };
00291
00292 #endif // SELENE_TYPES

```

## 6.16 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](#) ()

### Variables

- [Cfg config](#)

## 6.16.1 Detailed Description

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

## 6.16.2 Function Documentation

### 6.16.2.1 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.

### 6.16.2.2 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)"

### 6.16.2.3 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.

#### 6.16.2.4 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 <a href="#">GlobalModel</a> to print into the console.
--------------------	---

## 6.17 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](#) ()

## 6.17.1 Function Documentation

### 6.17.1.1 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 <a href="#">Agent</a> to parse into a string.
------------	---

#### Returns

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

### 6.17.1.2 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)"

### 6.17.1.3 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.

**6.17.1.4 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 <a href="#">GlobalModel</a> to print into the console.
--------------------	---

**6.18 Utils.hpp**

[Go to the documentation of this file.](#)

```
00001
00005 #ifndef STV_TYPES
00006 #define STV_TYPES
00007
00008 #include "Types.hpp"
00009 #include "Constants.hpp"
00010 #include <map>
00011 #include <string>
00012 #include <unistd.h>
00013 #include <sys/time.h>
00014 #include <iostream>
00015 #include <fstream>
00016
00017 using namespace std;
00018
00019 string envToString(map<string, int> env);
00020 string agentToString(Agent* agt);
00021 string localModelsToString(LocalModels* lm);
00022 void outputGlobalModel(GlobalModel* globalModel);
00023 unsigned long getMemCap();
00024
00025 #endif // STV_TYPES
```

**6.19 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

### 6.19.1 Detailed Description

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

### 6.19.2 Function Documentation

#### 6.19.2.1 `dbgHistEnt()`

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

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

##### Parameters

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

#### 6.19.2.2 `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 <a href="#">GlobalState</a> .
<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".

**6.19.2.3 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.

**6.20 Verification.hpp File Reference**

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

**Classes**

- struct [HistoryEntry](#)  
*Structure used to save model traversal history.*
- class [HistoryDbg](#)
- class [Verification](#)

**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.*

### 6.20.1 Enumeration Type Documentation

#### 6.20.1.1 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.

#### 6.20.1.2 Mode

enum [Mode](#)

Current model traversal mode.

Enumerator

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

### 6.20.2 Function Documentation

#### 6.20.2.1 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.

## 6.21 Verification.hpp

[Go to the documentation of this file.](#)

```

00001
00005 #ifndef SELENE_VERIFICATION
00006 #define SELENE_VERIFICATION
00007
00008 #include <stack>
00009 #include "Types.hpp"
00010 #include "GlobalModelGenerator.hpp"
00011 #include "SeleneFormula.hpp"
00012
00013 string verStatusToStr(GlobalStateVerificationStatus status);
00014
00016 enum HistoryEntryType {
00017     DECISION,
00018     STATE_STATUS,
00019     CONTEXT,
00020     MARK_DECISION_AS_INVALID,
00021 };
00022
00024 struct HistoryEntry {
00026     HistoryEntryType type;
00028     GlobalState* globalState;
00030     GlobalTransition* decision;
00032     bool globalTransitionControlled;
00034     GlobalStateVerificationStatus prevStatus;
00036     GlobalStateVerificationStatus newStatus;
00038     int depth;
00040     HistoryEntry* prev;
00042     HistoryEntry* next;
00044     string toString() {
00046         char buff[1024] = { 0 };
00047         if (this->type == HistoryEntryType::DECISION) {
00048             snprintf(buff, sizeof(buff), "decision in %s: to %s", this->globalState->hash.c_str(),
this->decision->to->hash.c_str());
00049         }
00050         else if (this->type == HistoryEntryType::STATE_STATUS) {
00051             snprintf(buff, sizeof(buff), "stateVerStatus of %s: %s -> %s",
this->globalState->hash.c_str(), verStatusToStr(this->prevStatus).c_str(),
verStatusToStr(this->newStatus).c_str());
00052         }
00053         else if (this->type == HistoryEntryType::CONTEXT) {
00054             snprintf(buff, sizeof(buff), "context in %s at depth %i: to %s (%s)",
this->globalState->hash.c_str(), this->depth, this->decision->to->hash.c_str(),
this->globalTransitionControlled ? "controlled" : "uncontrolled");
00055         }
00056         else if (this->type == HistoryEntryType::MARK_DECISION_AS_INVALID) {
00057             snprintf(buff, sizeof(buff), "markInvalid in %s: to %s", this->globalState->hash.c_str(),
this->decision->to->hash.c_str());
00058         }
00059         return string(buff);
00060     };
00061 };
00062
00063 class HistoryDbg {
00064 public:
00066     vector<pair<HistoryEntry*, char>> entries;
00067     HistoryDbg();
00068     ~HistoryDbg();
00069     void addEntry(HistoryEntry* entry);
00070     void markEntry(HistoryEntry* entry, char chr);
00071     void print(string prefix);
00072     HistoryEntry* cloneEntry(HistoryEntry* entry);
00073 };
00074
00075 // On-the-fly traversal mode
00077 enum Mode {
00078     NORMAL,

```

```

00079     REVERT,
00080     RESTORE,
00081 };
00082
00083 class Verification {
00084 public:
00085     Verification(GlobalModelGenerator* generator);
00086     ~Verification();
00087     bool verify();
00088 protected:
00089     Mode mode;
00090     GlobalState* revertToGlobalState;
00091     stack<HistoryEntry*> historyToRestore;
00092     GlobalModelGenerator* generator;
00093     SeleneFormula* seleneFormula;
00094     HistoryEntry* historyStart;
00095     HistoryEntry* historyEnd;
00096     bool verifyLocalStates(set<LocalState*>* localStates);
00097     bool verifyGlobalState(GlobalState* globalState, int depth);
00098     bool isGlobalTransitionControlledByCoalition(GlobalTransition* globalTransition);
00099     bool isAgentInCoalition(Agent* agent);
00100     EpistemicClass* getEpistemicClassForGlobalState(GlobalState* globalState);
00101     bool areGlobalStatesInTheSameEpistemicClass(GlobalState* globalState1, GlobalState* globalState2);
00102     void addHistoryDecision(GlobalState* globalState, GlobalTransition* decision);
00103     void addHistoryStateStatus(GlobalState* globalState, GlobalStateVerificationStatus prevStatus,
00104                               GlobalStateVerificationStatus newStatus);
00105     void addHistoryContext(GlobalState* globalState, int depth, GlobalTransition* decision, bool
00106                           globalTransitionControlled);
00107     void addHistoryMarkDecisionAsInvalid(GlobalState* globalState, GlobalTransition* decision);
00108     HistoryEntry* newHistoryMarkDecisionAsInvalid(GlobalState* globalState, GlobalTransition*
00109                                                   decision);
00110     bool revertLastDecision(int depth);
00111     void undoLastHistoryEntry(bool freeMemory);
00112     void undoHistoryUntil(HistoryEntry* historyEntry, bool inclusive, int depth);
00113     void printCurrentHistory(int depth);
00114 };
00115
00116 #endif // SELENE_VERIFICATION

```



# Index

- addEntry
  - HistoryDbg, [36](#)
- addHistoryContext
  - Verification, [46](#)
- addHistoryDecision
  - Verification, [47](#)
- addHistoryMarkDecisionAsInvalid
  - Verification, [47](#)
- addHistoryStateStatus
  - Verification, [47](#)
- addInitial
  - AgentTemplate, [11](#)
- addLocal
  - AgentTemplate, [11](#)
- addPersistent
  - AgentTemplate, [12](#)
- addTransition
  - AgentTemplate, [12](#)
- Agent, [9](#)
  - Agent, [10](#)
  - includesState, [10](#)
- AgentTemplate, [10](#)
  - addInitial, [11](#)
  - addLocal, [11](#)
  - addPersistent, [12](#)
  - addTransition, [12](#)
  - generateAgent, [12](#)
  - setId, [13](#)
  - setInitState, [13](#)
- agentToString
  - Utils.cpp, [67](#)
  - Utils.hpp, [69](#)
- areGlobalStatesInTheSameEpistemicClass
  - Verification, [48](#)
- Assignment, [14](#)
  - Assignment, [14](#)
- Cfg, [15](#)
- checkLocalTransitionConditions
  - GlobalModelGenerator, [30](#)
- cloneEntry
  - HistoryDbg, [37](#)
- compare
  - LocalState, [40](#)
- computeEpistemicClassHash
  - GlobalModelGenerator, [31](#)
- computeGlobalStateHash
  - GlobalModelGenerator, [31](#)
- Condition, [15](#)
- ConditionOperator
  - Types.hpp, [63](#)
- Constants.hpp, [53](#)
- CONTEXT
  - Verification.hpp, [73](#)
- dbgHistEnt
  - Verification.cpp, [71](#)
- dbgVerifStatus
  - Verification.cpp, [71](#)
- DECISION
  - Verification.hpp, [73](#)
- envToString
  - Utils.cpp, [67](#)
  - Utils.hpp, [69](#)
- EpistemicClass, [15](#)
- Equals
  - Types.hpp, [63](#)
- eval
  - ExprAdd, [16](#)
  - ExprAnd, [17](#)
  - ExprConst, [18](#)
  - ExprDiv, [18](#)
  - ExprEq, [19](#)
  - ExprGe, [20](#)
  - ExprGt, [21](#)
  - ExprIdent, [21](#)
  - ExprLe, [22](#)
  - ExprLt, [23](#)
  - ExprMul, [24](#)
  - ExprNe, [24](#)
  - ExprNot, [26](#)
  - ExprOr, [26](#)
  - ExprRem, [27](#)
  - ExprSub, [28](#)
- expandState
  - GlobalModelGenerator, [31](#)
- ExprAdd, [16](#)
  - eval, [16](#)
- ExprAnd, [17](#)
  - eval, [17](#)
- ExprConst, [17](#)
  - eval, [18](#)
- ExprDiv, [18](#)
  - eval, [18](#)
- ExprEq, [19](#)
  - eval, [19](#)
- expressions.cc, [54](#)
- expressions.hpp, [55](#), [56](#)
- ExprGe, [20](#)

- eval, [20](#)
- ExprGt, [20](#)
  - eval, [21](#)
- ExprIdent, [21](#)
  - eval, [21](#)
- ExprLe, [22](#)
  - eval, [22](#)
- ExprLt, [23](#)
  - eval, [23](#)
- ExprMul, [23](#)
  - eval, [24](#)
- ExprNe, [24](#)
  - eval, [24](#)
- ExprNode, [25](#)
- ExprNot, [25](#)
  - eval, [26](#)
- ExprOr, [26](#)
  - eval, [26](#)
- ExprRem, [27](#)
  - eval, [27](#)
- ExprSub, [28](#)
  - eval, [28](#)
- findGlobalStateInEpistemicClass
  - GlobalModelGenerator, [32](#)
- findOrCreateEpistemicClass
  - GlobalModelGenerator, [32](#)
- Formula, [28](#)
- FromValue
  - Types.hpp, [63](#)
- FromVar
  - Types.hpp, [63](#)
- generateAgent
  - AgentTemplate, [12](#)
- generateGlobalTransitions
  - GlobalModelGenerator, [32](#)
- generateInitState
  - GlobalModelGenerator, [33](#)
- generateStateFromLocalStates
  - GlobalModelGenerator, [33](#)
- getCurrentGlobalModel
  - GlobalModelGenerator, [33](#)
- getEpistemicClassForGlobalState
  - Verification, [48](#)
- getFormula
  - GlobalModelGenerator, [34](#)
- GlobalModel, [29](#)
- GlobalModelGenerator, [29](#)
  - checkLocalTransitionConditions, [30](#)
  - computeEpistemicClassHash, [31](#)
  - computeGlobalStateHash, [31](#)
  - expandState, [31](#)
  - findGlobalStateInEpistemicClass, [32](#)
  - findOrCreateEpistemicClass, [32](#)
  - generateGlobalTransitions, [32](#)
  - generateInitState, [33](#)
  - generateStateFromLocalStates, [33](#)
  - getCurrentGlobalModel, [33](#)
  - getFormula, [34](#)
  - initModel, [34](#)
  - GlobalModelGenerator.cpp, [53](#)
  - GlobalModelGenerator.hpp, [53](#), [54](#)
  - GlobalState, [34](#)
  - GlobalStateVerificationStatus
    - Types.hpp, [63](#)
  - GlobalTransition, [35](#)
  - HistoryDbg, [36](#)
    - addEntry, [36](#)
    - cloneEntry, [37](#)
    - markEntry, [37](#)
    - print, [37](#)
  - HistoryEntry, [38](#)
    - toString, [38](#)
  - HistoryEntryType
    - Verification.hpp, [73](#)
  - includesState
    - Agent, [10](#)
  - initModel
    - GlobalModelGenerator, [34](#)
  - isAgentInCoalition
    - Verification, [48](#)
  - isGlobalTransitionControlledByCoalition
    - Verification, [49](#)
  - LocalModels, [39](#)
  - localModelsToString
    - Utils.cpp, [67](#)
    - Utils.hpp, [69](#)
  - LocalState, [39](#)
    - compare, [40](#)
  - LocalStateTemplate, [40](#)
  - LocalTransition, [41](#)
  - MARK\_DECISION\_AS\_INVALID
    - Verification.hpp, [73](#)
  - markEntry
    - HistoryDbg, [37](#)
  - Mode
    - Verification.hpp, [73](#)
  - newHistoryMarkDecisionAsInvalid
    - Verification, [49](#)
  - nodes.cc, [58](#)
  - nodes.hpp, [59](#)
  - NORMAL
    - Verification.hpp, [73](#)
  - NotEquals
    - Types.hpp, [63](#)
  - outputGlobalModel
    - Utils.cpp, [68](#)
    - Utils.hpp, [70](#)
  - parse
    - TestParser, [43](#)
  - PENDING

- Types.hpp, 63
- print
  - HistoryDbg, 37
- printCurrentHistory
  - Verification, 50
- RESTORE
  - Verification.hpp, 73
- REVERT
  - Verification.hpp, 73
- revertLastDecision
  - Verification, 50
- SeleneFormula, 42
- SeleneFormula.hpp, 60
- SeleneFormula1, 42
  - verifyLocalStates, 42
- setIdent
  - AgentTemplate, 13
- setInitState
  - AgentTemplate, 13
- STATE\_STATUS
  - Verification.hpp, 73
- TestParser, 43
  - parse, 43
- TestParser.hpp, 61
- toString
  - HistoryEntry, 38
- TransitionTemplate, 43
- Types.cc, 61
- Types.hpp, 62, 63
  - ConditionOperator, 63
  - Equals, 63
  - FromValue, 63
  - FromVar, 63
  - GlobalStateVerificationStatus, 63
  - NotEquals, 63
  - PENDING, 63
  - UNVERIFIED, 63
  - VarAssignmentType, 63
  - VERIFIED\_OK, 63
- undoHistoryUntil
  - Verification, 50
- undoLastHistoryEntry
  - Verification, 51
- UNVERIFIED
  - Types.hpp, 63
- Utils.cpp, 66
  - agentToString, 67
  - envToString, 67
  - localModelsToString, 67
  - outputGlobalModel, 68
- Utils.hpp, 68, 70
  - agentToString, 69
  - envToString, 69
  - localModelsToString, 69
  - outputGlobalModel, 70
- Var, 44
- VarAssignment, 44
- VarAssignmentType
  - Types.hpp, 63
- Verification, 45
  - addHistoryContext, 46
  - addHistoryDecision, 47
  - addHistoryMarkDecisionAsInvalid, 47
  - addHistoryStateStatus, 47
  - areGlobalStatesInTheSameEpistemicClass, 48
  - getEpistemicClassForGlobalState, 48
  - isAgentInCoalition, 48
  - isGlobalTransitionControlledByCoalition, 49
  - newHistoryMarkDecisionAsInvalid, 49
  - printCurrentHistory, 50
  - revertLastDecision, 50
  - undoHistoryUntil, 50
  - undoLastHistoryEntry, 51
  - Verification, 46
  - verify, 51
  - verifyGlobalState, 51
  - verifyLocalStates, 51
- Verification.cpp, 70
  - dbgHistEnt, 71
  - dbgVerifStatus, 71
  - verStatusToStr, 72
- Verification.hpp, 72, 74
  - CONTEXT, 73
  - DECISION, 73
  - HistoryEntryType, 73
  - MARK\_DECISION\_AS\_INVALID, 73
  - Mode, 73
  - NORMAL, 73
  - RESTORE, 73
  - REVERT, 73
  - STATE\_STATUS, 73
  - verStatusToStr, 73
- VERIFIED\_OK
  - Types.hpp, 63
- verify
  - Verification, 51
- verifyGlobalState
  - Verification, 51
- verifyLocalStates
  - SeleneFormula1, 42
  - Verification, 51
- verStatusToStr
  - Verification.cpp, 72
  - Verification.hpp, 73