

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
5.2 AgentTemplate Class Reference	9
5.3 Assignment Class Reference	10
5.4 Cfg Struct Reference	10
5.5 Condition Struct Reference	10
5.5.1 Detailed Description	11
5.6 EpistemicClass Struct Reference	11
5.7 ExprAdd Class Reference	11
5.7.1 Member Function Documentation	11
5.7.1.1 eval()	11
5.8 ExprAnd Class Reference	12
5.8.1 Member Function Documentation	12
5.8.1.1 eval()	12
5.9 ExprConst Class Reference	12
5.9.1 Member Function Documentation	12
5.9.1.1 eval()	12
5.10 ExprDiv Class Reference	13
5.10.1 Member Function Documentation	13
5.10.1.1 eval()	13
5.11 ExprEq Class Reference	13
5.11.1 Member Function Documentation	13
5.11.1.1 eval()	13
5.12 ExprGe Class Reference	14
5.12.1 Member Function Documentation	14
5.12.1.1 eval()	14
5.13 ExprGt Class Reference	14
5.13.1 Member Function Documentation	14
5.13.1.1 eval()	14
5.14 ExprIdent Class Reference	15
5.14.1 Member Function Documentation	15
5.14.1.1 eval()	15
5.15 ExprLe Class Reference	15

5.15.1 Member Function Documentation	15
5.15.1.1 eval()	15
5.16 ExprLt Class Reference	16
5.16.1 Member Function Documentation	16
5.16.1.1 eval()	16
5.17 ExprMul Class Reference	16
5.17.1 Member Function Documentation	16
5.17.1.1 eval()	16
5.18 ExprNe Class Reference	17
5.18.1 Member Function Documentation	17
5.18.1.1 eval()	17
5.19 ExprNode Class Reference	17
5.20 ExprNot Class Reference	17
5.20.1 Member Function Documentation	18
5.20.1.1 eval()	18
5.21 ExprOr Class Reference	18
5.21.1 Member Function Documentation	18
5.21.1.1 eval()	18
5.22 ExprRem Class Reference	19
5.22.1 Member Function Documentation	19
5.22.1.1 eval()	19
5.23 ExprSub Class Reference	19
5.23.1 Member Function Documentation	19
5.23.1.1 eval()	19
5.24 Formula Struct Reference	20
5.25 GlobalModel Struct Reference	20
5.26 GlobalModelGenerator Class Reference	20
5.27 GlobalState Struct Reference	21
5.28 GlobalTransition Struct Reference	21
5.29 HistoryDbg Class Reference	22
5.29.1 Member Function Documentation	22
5.29.1.1 addEntry()	22
5.29.1.2 cloneEntry()	22
5.29.1.3 markEntry()	23
5.29.1.4 print()	23
5.30 HistoryEntry Struct Reference	23
5.30.1 Detailed Description	24
5.30.2 Member Function Documentation	24
5.30.2.1 toString()	24
5.31 LocalModels Struct Reference	25
5.32 LocalState Class Reference	25
5.33 LocalStateTemplate Class Reference	25

5.34 LocalTransition Struct Reference . . . . .	26
5.35 SeleneFormula Class Reference . . . . .	26
5.36 SeleneFormula1 Class Reference . . . . .	26
5.36.1 Member Function Documentation . . . . .	27
5.36.1.1 verifyLocalStates() . . . . .	27
5.37 TestParser Class Reference . . . . .	27
5.38 TransitionTemplate Class Reference . . . . .	27
5.39 Var Struct Reference . . . . .	28
5.39.1 Detailed Description . . . . .	28
5.40 VarAssignment Struct Reference . . . . .	28
5.41 Verification Class Reference . . . . .	28
5.41.1 Constructor & Destructor Documentation . . . . .	30
5.41.1.1 Verification() . . . . .	30
5.41.2 Member Function Documentation . . . . .	30
5.41.2.1 addHistoryContext() . . . . .	30
5.41.2.2 addHistoryDecision() . . . . .	30
5.41.2.3 addHistoryMarkDecisionAsInvalid() . . . . .	31
5.41.2.4 addHistoryStateStatus() . . . . .	31
5.41.2.5 areGlobalStatesInTheSameEpistemicClass() . . . . .	31
5.41.2.6 getEpistemicClassForGlobalState() . . . . .	32
5.41.2.7 isAgentInCoalition() . . . . .	32
5.41.2.8 isGlobalTransitionControlledByCoalition() . . . . .	33
5.41.2.9 newHistoryMarkDecisionAsInvalid() . . . . .	33
5.41.2.10 printCurrentHistory() . . . . .	33
5.41.2.11 revertLastDecision() . . . . .	34
5.41.2.12 undoHistoryUntil() . . . . .	34
5.41.2.13 undoLastHistoryEntry() . . . . .	34
5.41.2.14 verify() . . . . .	35
5.41.2.15 verifyGlobalState() . . . . .	35
5.41.2.16 verifyLocalStates() . . . . .	35
<b>6 File Documentation . . . . .</b>	<b>37</b>
6.1 Constants.hpp . . . . .	37
6.2 GlobalModelGenerator.hpp . . . . .	37
6.3 expressions.hpp . . . . .	38
6.4 nodes.hpp . . . . .	40
6.5 SeleneFormula.hpp . . . . .	41
6.6 TestParser.hpp . . . . .	42
6.7 Types.hpp . . . . .	42
6.8 src/Utils.cpp File Reference . . . . .	44
6.8.1 Detailed Description . . . . .	45
6.8.2 Function Documentation . . . . .	45

---

6.8.2.1 agentToString()	45
6.8.2.2 envToString()	45
6.8.2.3 localModelsToString()	45
6.8.2.4 outputGlobalModel()	46
6.9 src/Utils.hpp File Reference	46
6.9.1 Function Documentation	47
6.9.1.1 agentToString()	47
6.9.1.2 envToString()	47
6.9.1.3 localModelsToString()	47
6.9.1.4 outputGlobalModel()	48
6.10 Utils.hpp	48
6.11 src/Verification.cpp File Reference	48
6.11.1 Detailed Description	49
6.11.2 Function Documentation	49
6.11.2.1 dbgHistEnt()	49
6.11.2.2 dbgVerifStatus()	49
6.11.2.3 verStatusToStr()	50
6.12 src/Verification.hpp File Reference	50
6.12.1 Enumeration Type Documentation	51
6.12.1.1 HistoryEntryType	51
6.12.1.2 Mode	51
6.12.2 Function Documentation	51
6.12.2.1 verStatusToStr()	51
6.13 Verification.hpp	52
<b>Index</b>	<b>55</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	9
Assignment	10
Cfg	10
Condition	10
EpistemicClass	11
ExprNode	17
ExprAdd	11
ExprAnd	12
ExprConst	12
ExprDiv	13
ExprEq	13
ExprGe	14
ExprGt	14
ExprIdent	15
ExprLe	15
ExprLt	16
ExprMul	16
ExprNe	17
ExprNot	17
ExprOr	18
ExprRem	19
ExprSub	19
Formula	20
GlobalModel	20
GlobalModelGenerator	20
GlobalState	21
GlobalTransition	21
HistoryDbg	22
HistoryEntry	23
LocalModels	25
LocalState	25
LocalStateTemplate	25
LocalTransition	26
SeleneFormula	26

SeleneFormula1 . . . . .	26
TestParser . . . . .	27
TransitionTemplate . . . . .	27
Var . . . . .	28
VarAssignment . . . . .	28
Verification . . . . .	28

## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">Agent</a>	9
<a href="#">AgentTemplate</a>	9
<a href="#">Assignment</a>	10
<a href="#">Cfg</a>	10
<a href="#">Condition</a>	
Represents condition	10
<a href="#">EpistemicClass</a>	11
<a href="#">ExprAdd</a>	11
<a href="#">ExprAnd</a>	12
<a href="#">ExprConst</a>	12
<a href="#">ExprDiv</a>	13
<a href="#">ExprEq</a>	13
<a href="#">ExprGe</a>	14
<a href="#">ExprGt</a>	14
<a href="#">ExprIdent</a>	15
<a href="#">ExprLe</a>	15
<a href="#">ExprLt</a>	16
<a href="#">ExprMul</a>	16
<a href="#">ExprNe</a>	17
<a href="#">ExprNode</a>	17
<a href="#">ExprNot</a>	17
<a href="#">ExprOr</a>	18
<a href="#">ExprRem</a>	19
<a href="#">ExprSub</a>	19
<a href="#">Formula</a>	20
<a href="#">GlobalModel</a>	20
<a href="#">GlobalModelGenerator</a>	20
<a href="#">GlobalState</a>	21
<a href="#">GlobalTransition</a>	21
<a href="#">HistoryDbg</a>	22
<a href="#">HistoryEntry</a>	
Structure used to save model traversal history	23
<a href="#">LocalModels</a>	25
<a href="#">LocalState</a>	25
<a href="#">LocalStateTemplate</a>	25

<a href="#">LocalTransition</a>	26
<a href="#">SeleneFormula</a>	26
<a href="#">SeleneFormula1</a>	26
<a href="#">TestParser</a>	27
<a href="#">TransitionTemplate</a>	27
<a href="#">Var</a>	
Represents variable in the model	28
<a href="#">VarAssignment</a>	28
<a href="#">Verification</a>	28

## Chapter 4

# File Index

### 4.1 File List

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

src/ <a href="#">Constants.hpp</a> . . . . .	37
src/ <a href="#">GlobalModelGenerator.hpp</a> . . . . .	37
src/ <a href="#">SeleneFormula.hpp</a> . . . . .	41
src/ <a href="#">TestParser.hpp</a> . . . . .	42
src/ <a href="#">Types.hpp</a> . . . . .	42
src/ <a href="#">Utils.cpp</a>	
Utility functions. A collection of utility functions to use in the project . . . . .	44
src/ <a href="#">Utils.hpp</a> . . . . .	46
src/ <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 . . . . .	48
src/ <a href="#">Verification.hpp</a> . . . . .	50
src/reader/ <a href="#">expressions.hpp</a> . . . . .	38
src/reader/ <a href="#">nodes.hpp</a> . . . . .	40



## Chapter 5

# Class Documentation

### 5.1 Agent Class Reference

#### Public Member Functions

- **Agent** (int \_id, string \_name)
- **LocalState** \* **includesState** (**LocalState** \*state)

#### Public Attributes

- int **id**
- string **name**
- set< **Var** \* > **vars**
- **LocalState** \* **initState**
- vector< **LocalState** \* > **localStates**
- vector< **LocalTransition** \* > **localTransitions**

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

- src/Types.hpp
- src/Types.cc

### 5.2 AgentTemplate Class Reference

#### Public Member Functions

- virtual **AgentTemplate** & **setIdent** (string \_ident)
- virtual **AgentTemplate** & **setInitState** (string \_startState)
- virtual **AgentTemplate** & **addLocal** (set< string > \*variables)
- virtual **AgentTemplate** & **addPersistent** (set< string > \*variables)
- virtual **AgentTemplate** & **addInitial** (set< **Assignment** \* > \*assigns)
- virtual **AgentTemplate** & **addTransition** (**TransitionTemplate** \*\_transition)
- virtual **Agent** \* **generateAgent** (int id)

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

- src/reader/nodes.hpp
- src/reader/nodes.cc

## 5.3 Assignment Class Reference

### Public Member Functions

- **Assignment** (string `_ident`, [ExprNode](#) \*`_exp`)
- virtual void **assign** (Environment &env)

### Public Attributes

- string **ident**
- [ExprNode](#) \* **value**

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

- src/reader/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:

- src/Constants.hpp

## 5.5 Condition Struct Reference

Represents condition.

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

### Public Attributes

- [Var](#) \* **var**
- ConditionOperator **conditionOperator**
- int **comparedValue**



### 5.5.1 Detailed Description

Represents condition.

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

- `src/Types.hpp`

## 5.6 EpistemicClass Struct Reference

### Public Attributes

- string **hash**
- map< string, [GlobalState](#) \* > **globalStates**
- [GlobalTransition](#) \* **fixedCoalitionTransition**

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

- `src/Types.hpp`

## 5.7 ExprAdd Class Reference

### Public Member Functions

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

### 5.7.1 Member Function Documentation

#### 5.7.1.1 eval()

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

Implements [ExprNode](#).

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

- `src/reader/expressions.hpp`
- `src/reader/expressions.cc`

## 5.8 ExprAnd Class Reference

### Public Member Functions

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

### 5.8.1 Member Function Documentation

#### 5.8.1.1 eval()

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

Implements [ExprNode](#).

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

- src/reader/expressions.hpp
- src/reader/expressions.cc

## 5.9 ExprConst Class Reference

### Public Member Functions

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

### 5.9.1 Member Function Documentation

#### 5.9.1.1 eval()

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

Implements [ExprNode](#).

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

- src/reader/expressions.hpp
- src/reader/expressions.cc

## 5.10 ExprDiv Class Reference

### Public Member Functions

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

### 5.10.1 Member Function Documentation

#### 5.10.1.1 eval()

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

Implements [ExprNode](#).

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

- src/reader/expressions.hpp
- src/reader/expressions.cc

## 5.11 ExprEq Class Reference

### Public Member Functions

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

### 5.11.1 Member Function Documentation

#### 5.11.1.1 eval()

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

Implements [ExprNode](#).

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

- src/reader/expressions.hpp
- src/reader/expressions.cc

## 5.12 ExprGe Class Reference

### Public Member Functions

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

### 5.12.1 Member Function Documentation

#### 5.12.1.1 eval()

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

Implements [ExprNode](#).

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

- src/reader/expressions.hpp
- src/reader/expressions.cc

## 5.13 ExprGt Class Reference

### Public Member Functions

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

### 5.13.1 Member Function Documentation

#### 5.13.1.1 eval()

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

Implements [ExprNode](#).

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

- src/reader/expressions.hpp
- src/reader/expressions.cc

## 5.14 ExprIdent Class Reference

### Public Member Functions

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

### 5.14.1 Member Function Documentation

#### 5.14.1.1 eval()

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

Implements [ExprNode](#).

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

- src/reader/expressions.hpp
- src/reader/expressions.cc

## 5.15 ExprLe Class Reference

### Public Member Functions

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

### 5.15.1 Member Function Documentation

#### 5.15.1.1 eval()

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

Implements [ExprNode](#).

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

- src/reader/expressions.hpp
- src/reader/expressions.cc

## 5.16 ExprLt Class Reference

### Public Member Functions

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

### 5.16.1 Member Function Documentation

#### 5.16.1.1 eval()

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

Implements [ExprNode](#).

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

- src/reader/expressions.hpp
- src/reader/expressions.cc

## 5.17 ExprMul Class Reference

### Public Member Functions

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

### 5.17.1 Member Function Documentation

#### 5.17.1.1 eval()

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

Implements [ExprNode](#).

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

- src/reader/expressions.hpp
- src/reader/expressions.cc

## 5.18 ExprNe Class Reference

### Public Member Functions

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

### 5.18.1 Member Function Documentation

#### 5.18.1.1 eval()

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

Implements [ExprNode](#).

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

- src/reader/expressions.hpp
- src/reader/expressions.cc

## 5.19 ExprNode Class Reference

### Public Member Functions

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

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

- src/reader/expressions.hpp

## 5.20 ExprNot Class Reference

### Public Member Functions

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

## 5.20.1 Member Function Documentation

### 5.20.1.1 eval()

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

Implements [ExprNode](#).

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

- src/reader/expressions.hpp
- src/reader/expressions.cc

## 5.21 ExprOr Class Reference

### Public Member Functions

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

## 5.21.1 Member Function Documentation

### 5.21.1.1 eval()

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

Implements [ExprNode](#).

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

- src/reader/expressions.hpp
- src/reader/expressions.cc



## 5.22 ExprRem Class Reference

### Public Member Functions

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

### 5.22.1 Member Function Documentation

#### 5.22.1.1 eval()

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

Implements [ExprNode](#).

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

- src/reader/expressions.hpp
- src/reader/expressions.cc

## 5.23 ExprSub Class Reference

### Public Member Functions

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

### 5.23.1 Member Function Documentation

#### 5.23.1.1 eval()

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

Implements [ExprNode](#).

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

- src/reader/expressions.hpp
- src/reader/expressions.cc

## 5.24 Formula Struct Reference

### Public Attributes

- set< [Agent](#) \* > **coalition**

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

- src/Types.hpp

## 5.25 GlobalModel Struct Reference

### Public Attributes

- vector< [Agent](#) \* > **agents**
- [Formula](#) \* **formula**
- [GlobalState](#) \* **initState**
- vector< [GlobalState](#) \* > **globalStates**
- vector< [GlobalTransition](#) \* > **globalTransitions**
- map< [Agent](#) \*, map< string, [EpistemicClass](#) \* > > **epistemicClasses**

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

- src/Types.hpp

## 5.26 GlobalModelGenerator Class Reference

### Public Member Functions

- [GlobalState](#) \* **initModel** ([LocalModels](#) \*localModels, [Formula](#) \*formula)
- void **expandState** ([GlobalState](#) \*state)
- void **expandAllStates** ()
- [GlobalModel](#) \* **getCurrentGlobalModel** ()
- [Formula](#) \* **getFormula** ()

### Protected Member Functions

- [GlobalState](#) \* **generateInitState** ()
- [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)
- string **computeGlobalStateHash** (set< [LocalState](#) \* > \*localStates)
- [EpistemicClass](#) \* **findOrCreateEpistemicClass** (set< [LocalState](#) \* > \*localStates, [Agent](#) \*agent)
- [GlobalState](#) \* **findGlobalStateInEpistemicClass** (set< [LocalState](#) \* > \*localStates, [EpistemicClass](#) \*epistemicClass)

## Protected Attributes

- [LocalModels](#) \* **localModels**
- [Formula](#) \* **formula**
- [GlobalModel](#) \* **globalModel**

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

- src/GlobalModelGenerator.hpp
- src/GlobalModelGenerator.cpp

## 5.27 GlobalState Struct Reference

### Public Attributes

- int **id**
- string **hash**
- map< [Var](#) \*, int > **vars**
- map< [Agent](#) \*, [EpistemicClass](#) \* > **epistemicClasses**
- bool **isExpanded**
- GlobalStateVerificationStatus **verificationStatus**
- set< [GlobalTransition](#) \* > **globalTransitions**
- set< [LocalState](#) \* > **localStates**

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

- src/Types.hpp

## 5.28 GlobalTransition Struct Reference

### Public Attributes

- int **id**
- bool **isInvalidDecision**
- [GlobalState](#) \* **from**
- [GlobalState](#) \* **to**
- set< [LocalTransition](#) \* > **localTransitions**

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

- src/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:

- [src/Verification.hpp](#)
- [src/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:

- [src/Verification.hpp](#)

## 5.31 LocalModels Struct Reference

### Public Attributes

- vector< [Agent](#) \* > **agents**
- map< string, [Var](#) \* > **vars**

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

- src/Types.hpp

## 5.32 LocalState Class Reference

### Public Member Functions

- bool **compare** ([LocalState](#) \*state)

### Public Attributes

- int **id**
- string **name**
- map< [Var](#) \*, int > **vars**
- map< string, int > **environment**
- [Agent](#) \* **agent**
- set< [LocalTransition](#) \* > **localTransitions**

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

- src/Types.hpp
- src/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:

- src/reader/nodes.hpp

## 5.34 LocalTransition Struct Reference

### Public Attributes

- int **id**
- string **name**
- string **localName**
- bool **isShared**
- int **sharedCount**
- set< [Condition](#) \* > **conditions**
- set< [VarAssignment](#) \* > **varAssignments**
- [Agent](#) \* **agent**
- [LocalState](#) \* **from**
- [LocalState](#) \* **to**
- set< [LocalTransition](#) \* > **sharedLocalTransitions**

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

- src/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:

- src/SeleneFormula.hpp
- src/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:

- src/SeleneFormula.hpp
- src/SeleneFormula.cpp

## 5.37 TestParser Class Reference

### Public Member Functions

- [LocalModels](#) \* **parse** (string fileName)

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

- src/TestParser.hpp
- src/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:

- src/reader/nodes.hpp

## 5.39 Var Struct Reference

Represents variable in the model.

```
#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 variable in the model.

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

- src/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:

- src/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 globalTransitionControlled)  
Creates a `HistoryEntry` of the type `CONTEXT` and puts it to the top of the decision history.
- void `addHistoryMarkDecisionAsInvalid` (`GlobalState` \*globalState, `GlobalTransition` \*decision)  
Creates a `HistoryEntry` of the type `MARK_DECISION_AS_INVALID` and puts it to the top of the decision history.
- `HistoryEntry` \* `newHistoryMarkDecisionAsInvalid` (`GlobalState` \*globalState, `GlobalTransition` \*decision)  
Creates a `HistoryEntry` of the type `MARK_DECISION_AS_INVALID` and returns it.
- bool `revertLastDecision` (int depth)  
Reverts `GlobalState` and history to the previous decision state.
- void `undoLastHistoryEntry` (bool freeMemory)  
Removes the top entry of the history stack.
- void `undoHistoryUntil` (`HistoryEntry` \*historyEntry, bool inclusive, int depth)  
Rolls back the history entries up to the certain `HistoryEntry`.
- void `printCurrentHistory` (int depth)  
Prints current history to the console.

## Protected Attributes

- `Mode` `mode`  
Current mode of model traversal.
- `GlobalState` \* `revertToGlobalState`  
Current lowest global state?
- stack< `HistoryEntry` \* > `historyToRestore`  
???
- `GlobalModelGenerator` \* `generator`  
Holds current model and formula.
- `SeleneFormula` \* `seleneFormula`  
???
- `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:

- [src/Verification.hpp](#)
- [src/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.hpp

```
00001 #ifndef SELENE_GLOBAL_MODEL_GENERATOR
00002 #define SELENE_GLOBAL_MODEL_GENERATOR
00003
00004 #include "Constants.hpp"
00005 #include "Types.hpp"
00006
00007 using namespace std;
00008
00009 class GlobalModelGenerator {
00010 public:
00011     GlobalModelGenerator();
00012     ~GlobalModelGenerator();
00013     GlobalState* initModel(LocalModels* localModels, Formula* formula);
00014     void expandState(GlobalState* state);
00015     void expandAllStates();
00016     GlobalModel* getCurrentGlobalModel();
00017     Formula* getFormula();
00018
00019 protected:
00020     LocalModels* localModels;
00021     Formula* formula;
00022     GlobalModel* globalModel;
00023     GlobalState* generateInitState();
00024     GlobalState* generateStateFromLocalStates(set<LocalState*>* localStates, set<LocalTransition*>*
viaLocalTransitions, GlobalState* prevGlobalState);
00025     void generateGlobalTransitions(GlobalState* fromGlobalState, set<LocalTransition*>
localTransitions, map<Agent*, vector<LocalTransition*>> transitionsByAgent);
00026     bool checkLocalTransitionConditions(LocalTransition* localTransition, GlobalState* globalState);
```

```

00027     string computeEpistemicClassHash(set<LocalState*>* localStates, Agent* agent);
00028     string computeGlobalStateHash(set<LocalState*>* localStates);
00029     EpistemicClass* findOrCreateEpistemicClass(set<LocalState*>* localStates, Agent* agent);
00030     GlobalState* findGlobalStateInEpistemicClass(set<LocalState*>* localStates, EpistemicClass*
    epistemicClass);
00031 };
00032
00033 #endif // SELENE_GLOBAL_MODEL_GENERATOR

```

## 6.3 expressions.hpp

```

00001 #ifndef __EXPRESSIONS_H
00002 #define __EXPRESSIONS_H
00003
00004 #include <string>
00005 #include <map>
00006
00007 using namespace std;
00008
00009 typedef map<string, int> Environment;
00010
00011 // węzeł bazowy dla wyrażeń
00012 class ExprNode {
00013 public:
00014     // metoda do wyliczenia wartości wyrażenia zależna od typu węzła
00015     virtual int eval( Environment& env ) = 0;
00016 };
00017
00018 // węzeł dla stałej
00019 class ExprConst: public ExprNode {
00020 public:
00021     // argumenty
00022     int val;
00023
00024 public:
00025     ExprConst(int _val): val(_val) {};
00026     virtual int eval( Environment& env );
00027 };
00028
00029 // węzeł dla identyfikatora
00030 class ExprIdent: public ExprNode {
00031 public:
00032     // argumenty
00033     string ident;
00034
00035 public:
00036     ExprIdent(string _ident): ident(_ident) {};
00037     virtual int eval( Environment& env );
00038 };
00039
00040 // węzeł dla dodawań
00041 class ExprAdd: public ExprNode {
00042 public:
00043     // argumenty
00044     ExprNode *larg, *rarg;
00045
00046 public:
00047     ExprAdd(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00048     virtual int eval( Environment& env );
00049 };
00050
00051 // węzeł dla odejmowań
00052 class ExprSub: public ExprNode {
00053 public:
00054     // argumenty
00055     ExprNode *larg, *rarg;
00056
00057 public:
00058     ExprSub(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00059     virtual int eval( Environment& env );
00060 };
00061
00062 // węzeł dla mnożeń
00063 class ExprMul: public ExprNode {
00064 public:
00065     // argumenty
00066     ExprNode *larg, *rarg;
00067
00068 public:
00069     ExprMul(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00070     virtual int eval( Environment& env );
00071 };
00072

```

```

00073
00074 // węzeł dla dzielenia
00075 class ExprDiv: public ExprNode {
00076
00077     // argumenty
00078     ExprNode *larg, *rarg;
00079
00080 public:
00081     ExprDiv(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00082     virtual int eval( Environment& env );
00083 };
00084
00085 // węzeł dla reszty z dzielenia
00086 class ExprRem: public ExprNode {
00087
00088     // argumenty
00089     ExprNode *larg, *rarg;
00090
00091 public:
00092     ExprRem(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00093     virtual int eval( Environment& env );
00094 };
00095
00096 // węzeł dla operatora AND
00097 class ExprAnd: public ExprNode {
00098
00099     // argumenty
00100     ExprNode *larg, *rarg;
00101
00102 public:
00103     ExprAnd(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00104     virtual int eval( Environment& env );
00105 };
00106
00107 // węzeł dla operatora OR
00108 class ExprOr: public ExprNode {
00109
00110     // argumenty
00111     ExprNode *larg, *rarg;
00112
00113 public:
00114     ExprOr(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00115     virtual int eval( Environment& env );
00116 };
00117
00118 // węzeł dla operatora NOT
00119 class ExprNot: public ExprNode {
00120
00121     // argumenty
00122     ExprNode *arg;
00123
00124 public:
00125     ExprNot(ExprNode *_arg): arg(_arg) {};
00126     virtual int eval( Environment& env );
00127 };
00128
00129 // węzeł dla operatora "=="
00130 class ExprEq: public ExprNode {
00131
00132     // argumenty
00133     ExprNode *larg, *rarg;
00134
00135 public:
00136     ExprEq(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00137     virtual int eval( Environment& env );
00138 };
00139
00140 // węzeł dla operatora "!="
00141 class ExprNe: public ExprNode {
00142
00143     // argumenty
00144     ExprNode *larg, *rarg;
00145
00146 public:
00147     ExprNe(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00148     virtual int eval( Environment& env );
00149 };
00150
00151 // węzeł dla operatora "<"
00152 class ExprLt: public ExprNode {
00153
00154     // argumenty
00155     ExprNode *larg, *rarg;
00156
00157 public:
00158     ExprLt(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00159     virtual int eval( Environment& env );

```

```

00160 };
00161
00162 // węzeł dla operatora "<="
00163 class ExprLe: public ExprNode {
00164
00165     // argumenty
00166     ExprNode *larg, *rarg;
00167
00168     public:
00169     ExprLe(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00170     virtual int eval( Environment& env );
00171 };
00172
00173 // węzeł dla operatora ">"
00174 class ExprGt: public ExprNode {
00175
00176     // argumenty
00177     ExprNode *larg, *rarg;
00178
00179     public:
00180     ExprGt(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00181     virtual int eval( Environment& env );
00182 };
00183
00184 // węzeł dla operatora ">="
00185 class ExprGe: public ExprNode {
00186
00187     // argumenty
00188     ExprNode *larg, *rarg;
00189
00190     public:
00191     ExprGe(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00192     virtual int eval( Environment& env );
00193 };
00194
00195
00196 #endif

```

## 6.4 nodes.hpp

```

00001 #ifndef __NODES_H
00002 #define __NODES_H
00003
00004 #include <string>
00005 #include <set>
00006 #include <map>
00007 #include "expressions.hpp"
00008
00009 #include "../Types.hpp"
00010
00011 using namespace std;
00012
00013 /* Klasa reprezentująca przypisanie */
00014 class Assignment {
00015     public:
00016     string ident;    // do czego przypisujemy
00017     ExprNode *value; // co przypisujemy
00018
00019     Assignment(string _ident, ExprNode *_exp): ident(_ident), value(_exp) {};
00020
00021     // wykonaj przypisanie w danym środowisku
00022     virtual void assign(Environment &env) {
00023         env[ident]=value->eval(env);
00024     };
00025 };
00026
00027 /* Klasa reprezentująca meta-tranzycję */
00028 class TransitionTemplate {
00029     public:
00030     // jeśli -1 to nie ma dzielenia, w p.p. wartość określa łączną liczbę wymaganych agentów
00031     int shared;
00032     // nazwa wzorca
00033     string patternName;
00034     // nazwa do wyszukiwania dla shared
00035     string matchName;
00036     // nazwa stanu początkowego i końcowego
00037     string startState;
00038     string endState;
00039     // wyrażenie warunkowe
00040     ExprNode *condition;
00041     // lista przypisań wartości
00042     set<Assignment*> *assignments;
00043

```

```

00044     TransitionTemplate(int _shared, string _patternName, string _matchName, string _startState,
00045         string _endState, ExprNode *_cond, set<Assignment*> *_assign):
00046         shared(_shared), patternName(_patternName), matchName(_matchName),
00047         startState(_startState), endState(_endState), condition(_cond), assignments(_assign) {};
00048 };
00049
00050 class LocalStateTemplate {
00051 public:
00052     string name;
00053     set<TransitionTemplate*> transitions;
00054 };
00055
00056 /* Klasa reprezentująca pojedynczego agenta po wczytaniu jego opisu z pliku */
00057 class AgentTemplate {
00058     // identyfikator agenta
00059     string ident;
00060     // stan startowy
00061     string initState;
00062     // zbiór zmiennych lokalnych (local)
00063     set<string*> localVars;
00064     // zbiór zmiennych trwałych (persistent)
00065     set<string*> persistentVars;
00066     // początkowa inicjacja
00067     set<Assignment*> initialAssignments;
00068     // zbiór tranzycji
00069     set<TransitionTemplate*> transitions;
00070
00071     // mapa stanów lokalnych potrzebna do wygenerowania modelu
00072     map<string, LocalStateTemplate*> localStateTemplates;
00073
00074     // metoda wyznaczająca węzeł kolejny do danego, zależnie od tranzycji
00075     virtual LocalState* genNextState(LocalState *state, TransitionTemplate *trans);
00076
00077 public:
00078     AgentTemplate();
00079
00080     // ustaw identyfikator agenta
00081     virtual AgentTemplate& setIdent(string _ident);
00082     // ustaw identyfikator agenta
00083     virtual AgentTemplate& setInitState(string _startState);
00084     // dodaj zmienna/zmienne lokalne
00085     virtual AgentTemplate& addLocal(set<string> *variables);
00086     // dodaj zmienne trwałe
00087     virtual AgentTemplate& addPersistent(set<string> *variables);
00088     // dodaj początkowe inicjacje
00089     virtual AgentTemplate& addInitial(set<Assignment*> *assigns);
00090     // dodaj tranzycję
00091     virtual AgentTemplate& addTransition(TransitionTemplate *_transition);
00092
00093     // wygeneruj agenta do modelu
00094     virtual Agent* generateAgent(int id) ;
00095 };
00096
00097 #endif

```

## 6.5 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

```

## 6.6 TestParser.hpp

```

00001 #ifndef __TESTPARSER_HPP
00002 #define __TESTPARSER_HPP
00003
00004 #include "Types.hpp"
00005
00006 using namespace std;
00007
00008 class TestParser {
00009 public:
00010     TestParser();
00011     ~TestParser();
00012     LocalModels* parse(string fileName);
00013
00014 protected:
00015     // @internal
00016 };
00017
00018 #endif

```

## 6.7 Types.hpp

```

00001 #ifndef SELENE_TYPES
00002 #define SELENE_TYPES
00003
00004 #include <map>
00005 #include <set>
00006 #include <stack>
00007 #include <string>
00008 #include <utility>
00009 #include <vector>
00010
00011 using namespace std;
00012
00013
00014 class Agent;
00015 class LocalState;
00016
00017 struct Condition;
00018 struct EpistemicClass;
00019 struct Formula;
00020 struct GlobalModel;
00021 struct GlobalState;
00022 struct GlobalTransition;
00023 struct LocalTransition;
00024 struct LocalModels;
00025 struct Var;
00026 struct VarAssignment;
00027
00028 enum ConditionOperator {
00029     Equals,
00030     NotEquals,
00031 };
00032
00033 enum GlobalStateVerificationStatus {
00034     UNVERIFIED,
00035     PENDING,
00036     VERIFIED_OK,
00037     VERIFIED_ERR,
00038 };
00039
00040 enum VarAssignmentType {
00041     FromVar,
00042     FromValue,
00043 };
00044

```



```

00046 struct Var {
00047     string name;
00048
00049     int initialValue;
00050
00051     bool persistent;
00052
00053     Agent *agent;
00054 };
00055
00056 struct Condition {
00057     Var* var;
00058     ConditionOperator conditionOperator;
00059     int comparedValue;
00060 };
00061
00062 struct Formula {
00063     set<Agent*> coalition;
00064 };
00065
00066 class Agent {
00067 public:
00068     int id;
00069     string name;
00070     set<Var*> vars;
00071     Agent(int _id, string _name):id(_id), name(_name) {};
00072
00073     LocalState* initState;
00074     vector<LocalState*> localStates; // localStates[i].id == i
00075     vector<LocalTransition*> localTransitions; // localTransitions[i].id == i
00076
00077     // sprawdź, czy stan nie został już wygenerowany
00078     LocalState* includesState(LocalState *state);
00079 };
00080
00081 class LocalState {
00082 public:
00083     // Data
00084     int id;
00085     string name;
00086     map<Var*, int> vars;
00087     // alternatywna wersja - może wystarczy
00088     map<string, int> environment;
00089
00090     // komparator
00091     bool compare(LocalState *state);
00092
00093     // Bindings
00094     Agent* agent;
00095     set<LocalTransition*> localTransitions;
00096 };
00097
00098 // to jest zbędne
00099 struct VarAssignment {
00100     Var* dstVar;
00101     VarAssignmentType type; // zbędne
00102     Var* srcVar;
00103     int value;
00104 };
00105
00106 struct LocalTransition {
00107     // Data
00108     int id;
00109     string name;
00110     string localName; // if empty => same as name
00111     bool isShared;
00112     int sharedCount;
00113     set<Condition*> conditions;
00114     set<VarAssignment*> varAsssignments;
00115
00116     // Bindings
00117     Agent* agent;
00118     LocalState* from;
00119     LocalState* to;
00120     set<LocalTransition*> sharedLocalTransitions;
00121 };
00122
00123 struct LocalModels {
00124     vector<Agent*> agents; // agents[i].id == i
00125     map<string, Var*> vars; // vars[str].name == str
00126 };
00127
00128 struct GlobalModel {
00129     // Data
00130     vector<Agent*> agents; // agents[i].id == i
00131     Formula* formula;
00132 };

```

```

00138     // Bindings
00139     GlobalState* initState;
00140     vector<GlobalState*> globalStates; // globalStates[i].id == i
00141     vector<GlobalTransition*> globalTransitions; // globalTransitions[i].id == i
00142     map<Agent*, map<string, EpistemicClass*>> epistemicClasses; // Agent* => (EpistemicClass->hash =>
    EpistemicClass*)
00143 };
00144
00145 struct GlobalState {
00146     // Data
00147     int id;
00148     string hash;
00149     map<Var*, int> vars;
00150     map<Agent*, EpistemicClass*> epistemicClasses;
00151     bool isExpanded;
00152     GlobalStateVerificationStatus verificationStatus;
00153
00154     // Bindings
00155     set<GlobalTransition*> globalTransitions;
00156     set<LocalState*> localStates;
00157 };
00158
00159 struct GlobalTransition {
00160     // Data
00161     int id;
00162     bool isInvalidDecision;
00163
00164     // Bindings
00165     GlobalState* from;
00166     GlobalState* to;
00167     set<LocalTransition*> localTransitions;
00168 };
00169
00170 struct EpistemicClass {
00171     string hash;
00172     map<string, GlobalState*> globalStates; // GlobalState->hash => GlobalState*
00173     GlobalTransition* fixedCoalitionTransition;
00174 };
00175
00176 #endif // SELENE_TYPES

```

## 6.8 src/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](#)

## 6.8.1 Detailed Description

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

## 6.8.2 Function Documentation

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

#### Returns

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

### 6.8.2.2 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)"

### 6.8.2.3 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.

#### 6.8.2.4 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.
--------------------	---

## 6.9 src/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](#) ()

## 6.9.1 Function Documentation

### 6.9.1.1 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.

### 6.9.1.2 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)"

### 6.9.1.3 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.

**6.9.1.4 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.
--------------------	---

**6.10 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
```

**6.11 src/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

### 6.11.1 Detailed Description

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

### 6.11.2 Function Documentation

#### 6.11.2.1 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 <a href="#">HistoryEntry</a> struct which will be printed out.

#### 6.11.2.2 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".

**6.11.2.3 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.

**6.12 src/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.*

### 6.12.1 Enumeration Type Documentation

#### 6.12.1.1 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.

#### 6.12.1.2 Mode

enum [Mode](#)

Current model traversal mode.

Enumerator

NORMAL	Normal model traversal.
REVERT	???
RESTORE	???

### 6.12.2 Function Documentation

#### 6.12.2.1 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.

## 6.13 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, [22](#)
- addHistoryContext
  - Verification, [30](#)
- addHistoryDecision
  - Verification, [30](#)
- addHistoryMarkDecisionAsInvalid
  - Verification, [31](#)
- addHistoryStateStatus
  - Verification, [31](#)
- Agent, [9](#)
- AgentTemplate, [9](#)
- agentToString
  - Utils.cpp, [45](#)
  - Utils.hpp, [47](#)
- areGlobalStatesInTheSameEpistemicClass
  - Verification, [31](#)
- Assignment, [10](#)
- Cfg, [10](#)
- cloneEntry
  - HistoryDbg, [22](#)
- Condition, [10](#)
- CONTEXT
  - Verification.hpp, [51](#)
- dbgHistEnt
  - Verification.cpp, [49](#)
- dbgVerifStatus
  - Verification.cpp, [49](#)
- DECISION
  - Verification.hpp, [51](#)
- envToString
  - Utils.cpp, [45](#)
  - Utils.hpp, [47](#)
- EpistemicClass, [11](#)
- eval
  - ExprAdd, [11](#)
  - ExprAnd, [12](#)
  - ExprConst, [12](#)
  - ExprDiv, [13](#)
  - ExprEq, [13](#)
  - ExprGe, [14](#)
  - ExprGt, [14](#)
  - ExprIdent, [15](#)
  - ExprLe, [15](#)
  - ExprLt, [16](#)
  - ExprMul, [16](#)
  - ExprNe, [17](#)
  - ExprNot, [18](#)
  - ExprOr, [18](#)
  - ExprRem, [19](#)
  - ExprSub, [19](#)
- ExprAdd, [11](#)
  - eval, [11](#)
- ExprAnd, [12](#)
  - eval, [12](#)
- ExprConst, [12](#)
  - eval, [12](#)
- ExprDiv, [13](#)
  - eval, [13](#)
- ExprEq, [13](#)
  - eval, [13](#)
- ExprGe, [14](#)
  - eval, [14](#)
- ExprGt, [14](#)
  - eval, [14](#)
- ExprIdent, [15](#)
  - eval, [15](#)
- ExprLe, [15](#)
  - eval, [15](#)
- ExprLt, [16](#)
  - eval, [16](#)
- ExprMul, [16](#)
  - eval, [16](#)
- ExprNe, [17](#)
  - eval, [17](#)
- ExprNode, [17](#)
- ExprNot, [17](#)
  - eval, [18](#)
- ExprOr, [18](#)
  - eval, [18](#)
- ExprRem, [19](#)
  - eval, [19](#)
- ExprSub, [19](#)
  - eval, [19](#)
- Formula, [20](#)
- getEpistemicClassForGlobalState
  - Verification, [32](#)
- GlobalModel, [20](#)
- GlobalModelGenerator, [20](#)
- GlobalState, [21](#)
- GlobalTransition, [21](#)
- HistoryDbg, [22](#)
  - addEntry, [22](#)
  - cloneEntry, [22](#)

- markEntry, 23
- print, 23
- HistoryEntry, 23
  - toString, 24
- HistoryEntryType
  - Verification.hpp, 51
- isAgentInCoalition
  - Verification, 32
- isGlobalTransitionControlledByCoalition
  - Verification, 32
- LocalModels, 25
- localModelsToString
  - Utils.cpp, 45
  - Utils.hpp, 47
- LocalState, 25
- LocalStateTemplate, 25
- LocalTransition, 26
- MARK\_DECISION\_AS\_INVALID
  - Verification.hpp, 51
- markEntry
  - HistoryDbg, 23
- Mode
  - Verification.hpp, 51
- newHistoryMarkDecisionAsInvalid
  - Verification, 33
- NORMAL
  - Verification.hpp, 51
- outputGlobalModel
  - Utils.cpp, 46
  - Utils.hpp, 48
- print
  - HistoryDbg, 23
- printCurrentHistory
  - Verification, 33
- RESTORE
  - Verification.hpp, 51
- REVERT
  - Verification.hpp, 51
- revertLastDecision
  - Verification, 33
- SeleneFormula, 26
- SeleneFormula1, 26
  - verifyLocalStates, 27
- src/Constants.hpp, 37
- src/GlobalModelGenerator.hpp, 37
- src/reader/expressions.hpp, 38
- src/reader/nodes.hpp, 40
- src/SeleneFormula.hpp, 41
- src/TestParser.hpp, 42
- src/Types.hpp, 42
- src/Utils.cpp, 44
- src/Utils.hpp, 46, 48
- src/Verification.cpp, 48
- src/Verification.hpp, 50, 52
- STATE\_STATUS
  - Verification.hpp, 51
- TestParser, 27
- toString
  - HistoryEntry, 24
- TransitionTemplate, 27
- undoHistoryUntil
  - Verification, 34
- undoLastHistoryEntry
  - Verification, 34
- Utils.cpp
  - agentToString, 45
  - envToString, 45
  - localModelsToString, 45
  - outputGlobalModel, 46
- Utils.hpp
  - agentToString, 47
  - envToString, 47
  - localModelsToString, 47
  - outputGlobalModel, 48
- Var, 28
- VarAssignment, 28
- Verification, 28
  - addHistoryContext, 30
  - addHistoryDecision, 30
  - addHistoryMarkDecisionAsInvalid, 31
  - addHistoryStateStatus, 31
  - areGlobalStatesInTheSameEpistemicClass, 31
  - getEpistemicClassForGlobalState, 32
  - isAgentInCoalition, 32
  - isGlobalTransitionControlledByCoalition, 32
  - newHistoryMarkDecisionAsInvalid, 33
  - printCurrentHistory, 33
  - revertLastDecision, 33
  - undoHistoryUntil, 34
  - undoLastHistoryEntry, 34
  - Verification, 30
  - verify, 34
  - verifyGlobalState, 35
  - verifyLocalStates, 35
- Verification.cpp
  - dbgHistEnt, 49
  - dbgVerifStatus, 49
  - verStatusToStr, 50
- Verification.hpp
  - CONTEXT, 51
  - DECISION, 51
  - HistoryEntryType, 51
  - MARK\_DECISION\_AS\_INVALID, 51
  - Mode, 51
  - NORMAL, 51
  - RESTORE, 51
  - REVERT, 51
  - STATE\_STATUS, 51

- verStatusToStr, [51](#)
- verify
  - Verification, [34](#)
- verifyGlobalState
  - Verification, [35](#)
- verifyLocalStates
  - SeleneFormula1, [27](#)
  - Verification, [35](#)
- verStatusToStr
  - Verification.cpp, [50](#)
  - Verification.hpp, [51](#)