stv_v2

Generated by Doxygen 1.9.6

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:

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

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

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 argment.

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

_id	Identifier of the new agent.
_name	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 argment.

Parameters

state	A pointer to LocalState to be checked.

Returns

Returns a pointer to an equivalent LocalState if such exists, otherwise returns NULL.

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

- Types.hpp
- Types.cc

5.2 AgentTemplate Class Reference

Public Member Functions

- **AgentTemplate** ()

Constructor for an AgentTemplate.

- virtual AgentTemplate & setIdent (string _ident)

  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

assigns	???

Returns

???

5.2.1.2 addLocal()

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

Adds local variables to the agent.

Parameters

variables	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

variables	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

_transition	???

Returns

???

5.2.1.5 generateAgent()

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

Generates an agent for the model.

Parameters

id	Identifier of the new Agent.

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

_ident	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

_initState	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

_ident	To what we should assign a value.
_exp	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

localTransition	???
globalState	???

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

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

Returns

Returns a string with a hash.

5.26.1.3 computeGlobalStateHash()

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

Creates a hash from a set of LocalState.

Parameters

localStates	Pointer to a set of pointers of LocalState to turn into a hash.

Returns

Returns a string with a hash.

5.26.1.4 expandState()

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

???

Parameters

state	???

5.26.1.5 findGlobalStateInEpistemicClass()

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

???

Parameters

localStates	???
epistemicClass	???

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

localStates	Local states from agent.
agent	Agent 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

fromGlobalState	???
localTransitions	???
transitionsByAgent	???

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

localStates	???
viaLocalTransitions	???
prevGlobalState	???

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

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

Returns

Returns a pointer to initial state of the global model.

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

- GlobalModelGenerator.hpp
- GlobalModelGenerator.cpp

5.27 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 verifivation 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 degug 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

entry	A pointer to the HistoryEntry 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

entry	A pointer to a HistoryEntry 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 degug history with a char.

Parameters

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

5.29.1.4 print()

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

Prints every entry from the algorithm's path.

Parameters

prefix	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 descriprion 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

state	A pointer to LocalState to which this state should be compared to.

Returns

Returns true if the current LocalState is the same as the passed one, otherwise false.

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

- Types.hpp
- Types.cc

5.33 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 ∗ > **varAsssignments**

  ???
- 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

fileName	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

generator	Pointer to GlobalModelGenerator

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

globalState	Pointer to a GlobalState of the model.
depth	Depth of the recursion of the validation algorithm.
decision	Pointer to a transition GlobalTransition selected by the algorithm.
globalTransitionControlled	True if the GlobalTransition 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

globalState	Pointer to a GlobalState of the model.
decision	Pointer to a GlobalTransition 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

globalState	Pointer to a GlobalState of the model.
decision	Pointer to a transition GlobalTransition 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

globalState	Pointer to a GlobalState of the model.
prevStatus	Previous GlobalStateVerificationStatus to be logged.
newStatus	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

globalState1	Pointer to the first GlobalState.
globalState2	Pointer to the second GlobalState.

Returns

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

5.41.2.6 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

globalState	Pointer to a GlobalState of the model.

Returns

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

5.41.2.7 isAgentInCoalition()

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

Checks if the Agent is in a coalition based on the formula in a GlobalModelGenerator.

Parameters

agent	Pointer to an Agent 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

globalTransition	Pointer to a GlobalTransition 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

globalState	Pointer to a GlobalState of the model.
decision	Pointer to a transition GlobalTransition 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

depth	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

depth	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

historyEntry	Pointer to a HistoryEntry that the history has to be rolled back to.
inclusive	True if the rollback has to remove the specified entry too.
depth	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

freeMemory	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

globalState	Pointer to a GlobalState of the model.
depth	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

localStates	A pointer to a set of pointers to LocalState.

Returns

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

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

- Verification.hpp
- Verification.cpp

Chapter 6

File Documentation

6.1 Constants.hpp

```
#ifndef SELENE_CONSTANTS
#define SELENE_CONSTANTS

#define VERBOSE 0
// #define OUTPUT_LOCAL_MODELS 1
// #define OUTPUT_GLOBAL_MODEL 0 // warning: it will call expandAllStates()
// #define MODE 2 // 1 = only generate; 2 = verify

// Model id
// 1 = src/examples/trains/Trains.txt
// 2 = src/examples/ssvr/Selene_Select_Vote_Revoting_1v_1cv_3c_3rev_share.txt
// 3 = src/examples/svote/Simple_voting.txt
// #define MODEL_ID 1

struct Cfg{
    char* fname;
    char stv_mode;
    bool output_local_models;
    bool output_global_model;
    int model_id; // <-- this is temporary member (used in Verification.cpp for a hardcoded formula)
};

#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.

```
#ifndef SELENE_GLOBAL_MODEL_GENERATOR
#define SELENE_GLOBAL_MODEL_GENERATOR

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

using namespace std;

class GlobalModelGenerator {
public:
    GlobalModelGenerator();
    ~GlobalModelGenerator();
    GlobalState* initModel(LocalModels* localModels, Formula* formula);
    void expandState(GlobalState* state);
    void expandAllStates();
    GlobalModel* getCurrentGlobalModel();
    Formula* getFormula();

protected:
    LocalModels* localModels;
    Formula* formula;
    GlobalModel* globalModel;
    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);
};

#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.

```

#ifndef __EXPRESSIONS_H
#define __EXPRESSIONS_H

#include <string>
#include <map>

using namespace std;

typedef map<string, int> Environment;

// węzeł bazowy dla wyrażeń

class ExprNode {

   public:
      // metoda do wyliczenia wartości wyrażenia zależna od typu węzła
      virtual int eval( Environment& env ) = 0;
};

// węzeł dla stałej

class ExprConst: public ExprNode {

   // argumenty

   int val;

   public:
      ExprConst(int _val): val(_val) {};
      virtual int eval( Environment& env );
};

// węzeł dla identyfikatora

class ExprIdent: public ExprNode {

   // argumenty

   string ident;

   public:
      ExprIdent(string _ident): ident(_ident) {};
      virtual int eval( Environment& env );
};

// węzeł dla dodawań

class ExprAdd: public ExprNode {

   // argumenty

   ExprNode *larg, *rarg;

   public:
      ExprAdd(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
      virtual int eval( Environment& env );
};

// węzeł dla odejmowań

class ExprSub: public ExprNode {

   // argumenty

   ExprNode *larg, *rarg;

   public:
      ExprSub(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
      virtual int eval( Environment& env );
};

```

```
// węzeł dla mnożeń

class ExprMul: public ExprNode {

   // argumenty

   ExprNode *larg, *rarg;

   public:
      ExprMul(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
      virtual int eval( Environment& env );
};

// węzeł dla dzieleń

class ExprDiv: public ExprNode {

   // argumenty

   ExprNode *larg, *rarg;

   public:
      ExprDiv(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
      virtual int eval( Environment& env );
};

// węzeł dla reszty z dzielenia

class ExprRem: public ExprNode {

   // argumenty

   ExprNode *larg, *rarg;

   public:
      ExprRem(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
      virtual int eval( Environment& env );
};

// węzeł dla operatora AND

class ExprAnd: public ExprNode {

   // argumenty

   ExprNode *larg, *rarg;

   public:
      ExprAnd(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
      virtual int eval( Environment& env );
};

// węzeł dla operatora OR

class ExprOr: public ExprNode {

   // argumenty

   ExprNode *larg, *rarg;

   public:
      ExprOr(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
      virtual int eval( Environment& env );
};

// węzeł dla operatora NOT

class ExprNot: public ExprNode {

   // argumenty

   ExprNode *arg;

   public:
      ExprNot(ExprNode *_arg): arg(_arg) {};
      virtual int eval( Environment& env );
};

// węzeł dla operatora "=="

class ExprEq: public ExprNode {

   // argumenty

   ExprNode *larg, *rarg;

   public:
      ExprEq(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
      virtual int eval( Environment& env );
};

// węzeł dla operatora "!="

class ExprNe: public ExprNode {

   // argumenty

   ExprNode *larg, *rarg;

   public:
      ExprNe(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
      virtual int eval( Environment& env );
};

// węzeł dla operatora "<"

class ExprLt: public ExprNode {

   // argumenty

   ExprNode *larg, *rarg;

   public:
      ExprLt(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
      virtual int eval( Environment& env );
};

// węzeł dla operatora "<="

class ExprLe: public ExprNode {

   // argumenty

   ExprNode *larg, *rarg;

   public:
      ExprLe(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
      virtual int eval( Environment& env );
};

// węzeł dla operatora ">"

class ExprGt: public ExprNode {

   // argumenty

   ExprNode *larg, *rarg;

   public:
      ExprGt(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
      virtual int eval( Environment& env );
};

// węzeł dla operatora ">="

class ExprGe: public ExprNode {

   // argumenty

   ExprNode *larg, *rarg;

   public:
      ExprGe(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
      virtual int eval( Environment& env );
};


#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.

```

#ifndef __NODES_H
#define __NODES_H

#include <string>
#include <set>
#include <map>
#include "expressions.hpp"

#include "../Types.hpp"

using namespace std;

/* Klasa reprezentująca przypisanie */
class Assignment {
   public:
      string ident;
      // do czego przypisujemy

      ExprNode *value;
      // co przypisujemy

      Assignment(string _ident, ExprNode *_exp): ident(_ident), value(_exp) {};

      // wykonaj przypisanie w danym środowisku
      virtual void assign(Environment &env) {
         env[ident]=value->eval(env);
      };
};

/* Klasa reprezentująca meta-tranzycję */
class TransitionTemplate {
   public:
      // jeśli -1 to nie ma dzielenia, w p.p. wartość określa łączną liczbę wymaganych agentów
      int shared;
      // nazwa wzorca
      string patternName;
      // nazwa do wyszukiwania dla shared
      string matchName;
      // nazwa stanu początkowego i końcowego
      string startState;
      string endState;
      // wyrażenie warunkowe
      ExprNode *condition;
      // lista przypisań wartości
      set<Assignment*> *assignments;

      TransitionTemplate(int _shared, string _patternName, string _matchName, string _startState, string _endState, ExprNode *_cond, set<Assignment*> *_assign):
            shared(_shared), patternName(_patternName), matchName(_matchName),
            startState(_startState), endState(_endState), condition(_cond), assignments(_assign) {};

};

class LocalStateTemplate {
   public:
      string name;
      set<TransitionTemplate*> transitions;
};

/* Klasa reprezentująca pojedynczego agenta po wczytaniu jego opisu z pliku */
class AgentTemplate {
      // identyfikator agenta
      string ident;
      // stan startowy
      string initState;
      // zbiór zmiennych lokalnych (local)
      set<string>* localVars;
      // zbiór zmiennych trwałych (persistent)
      set<string>* persistentVars;
      // początkowa inicjacja
      set<Assignment*>* initialAssignments;
      // zbiór tranzycji
      set<TransitionTemplate*>* transitions;

      // mapa stanów lokalnych potrzebna do wygenerowania modelu
      map<string,LocalStateTemplate*> localStateTemplates;

      // metoda wyznaczająca węzeł kolejny do danego, zależnie od tranzycji
      virtual LocalState* genNextState(LocalState *state, TransitionTemplate *trans);

   public:
      AgentTemplate();

      // ustaw identyfikator agenta
      virtual AgentTemplate& setIdent(string _ident);
      // ustaw identyfikator agenta
      virtual AgentTemplate& setInitState(string _startState);
      // dodaj zmienną/zmienne lokalne
      virtual AgentTemplate& addLocal(set<string> *variables);
      // dodaj zmienne trwałe
      virtual AgentTemplate& addPersistent(set<string> *variables);
      // dodaj początkowe inicjacje
      virtual AgentTemplate& addInitial(set<Assignment*> *assigns);
      // dodaj tranzycję
      virtual AgentTemplate& addTransition(TransitionTemplate *_transition);

      // wygeneruj agenta do modelu
      virtual Agent* generateAgent(int id) ;
};

#endif
```

6.11 SeleneFormula.hpp

```
#ifndef SELENE_SELENE_FORMULA
#define SELENE_SELENE_FORMULA

#include "Types.hpp"


```

```



class SeleneFormula {
public:
    SeleneFormula();
    ~SeleneFormula();
    virtual bool verifyLocalStates(set<LocalState*>* localStates) = 0;
    LocalState* getLocalStateForAgent(string agentName, set<LocalState*>* localStates);
    int getLocalStateVar(string varName, LocalState* localState);
    inline bool implication(bool left, bool right);
};





class SeleneFormula1 : public SeleneFormula {
public:
    SeleneFormula1();
    ~SeleneFormula1();
    bool verifyLocalStates(set<LocalState*>* localStates);
protected:
};





#endif // SELENE_SELENE_FORMULA
```

6.12 TestParser.hpp

```

#ifndef __TESTPARSER_HPP
#define __TESTPARSER_HPP

#include "Types.hpp"

using namespace std;

class TestParser {
public:
    TestParser();
    ~TestParser();
    LocalModels* parse(string fileName);

protected:
    // @internal
};

#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.

```

#ifndef SELENE_TYPES
#define SELENE_TYPES

#include <map>
#include <set>

#include <stack>
#include <string>
#include <utility>
#include <vector>

using namespace std;


class Agent;
class LocalState;

struct Condition;
struct EpistemicClass;
struct Formula;
struct GlobalModel;
struct GlobalState;
struct GlobalTransition;
struct LocalTransition;
struct LocalModels;
struct Var;
struct VarAssignment;

enum ConditionOperator {
    Equals,
    NotEquals,
};

enum GlobalStateVerificationStatus {
    UNVERIFIED,
    PENDING,
    VERIFIED_OK,
    VERIFIED_ERR,
};

enum VarAssignmentType {
    FromVar,
    FromValue,
};

struct Var {
    string name;

    int initialValue;

    bool persistent;

    Agent *agent;
};

struct Condition {
    Var* var;

    ConditionOperator conditionOperator;

    int comparedValue;
};

struct Formula {
    set<Agent*> coalition;
};

class Agent {
    public:
        int id;

        string name;

        set<Var*> vars;

        Agent(int _id, string _name):id(_id), name(_name) {};

        LocalState* initState;

        vector<LocalState*> localStates; // localStates[i].id == i

        vector<LocalTransition*> localTransitions; // localTransitions[i].id == i

        // sprawdź, czy stan nie został już wygenerowany.
        LocalState* includesState(LocalState *state);
};
```

```cpp

class LocalState {
    public:
        // Data
        int id;

        string name;

        map<Var*, int> vars;

        // alternatywna wersja - może wystarczy
        map<string, int> environment;

        // komparator
        bool compare(LocalState *state);

        // Bindings
        Agent* agent;

        set<LocalTransition*> localTransitions;
};

// to jest zbędne
struct VarAssignment {
    Var* dstVar;
    VarAssignmentType type; // zbędne
    Var* srcVar;
    int value;
};

struct LocalTransition {
    // Data
    int id;

    string name;

    string localName; // if empty => same as name

    bool isShared;

    int sharedCount;

    set<Condition*> conditions;

    set<VarAssignment*> varAsssignments;

    // Bindings
    Agent* agent;

    LocalState* from;

    LocalState* to;

    set<LocalTransition*> sharedLocalTransitions;
};

struct LocalModels {
    vector<Agent*> agents;
    // agents[i].id == i

    map<string, Var*> vars;
    // vars[str].name == str
};

struct GlobalModel {
    // Data
    vector<Agent*> agents;
    // agents[i].id == i

    Formula* formula;

    // Bindings
    GlobalState* initState;

    vector<GlobalState*> globalStates;
    // globalStates[i].id == i

    vector<GlobalTransition*> globalTransitions;
    // globalTransitions[i].id == i

    map<Agent*, map<string, EpistemicClass*» epistemicClasses;
    // Agent* => (EpistemicClass->hash => EpistemicClass*)
};

struct GlobalState {
    // Data
    int id;
    string hash;

    map<Var*, int> vars;

    map<Agent*, EpistemicClass*> epistemicClasses;

    bool isExpanded;

    GlobalStateVerificationStatus verificationStatus;

    // Bindings
    set<GlobalTransition*> globalTransitions;

    set<LocalState*> localStates;
};

struct GlobalTransition {
    // Data
    int id;

    bool isInvalidDecision;

    // Bindings
    GlobalState* from;

    GlobalState* to;

    set<LocalTransition*> localTransitions;
};

struct EpistemicClass {
    string hash;

    map<string, GlobalState*> globalStates;
    // GlobalState->hash => GlobalState*

    GlobalTransition* fixedCoalitionTransition;
};

#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 Agant 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 cointaining 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 Agant into a string containing name of the agent, its initial state, transitions with their local and global names, shared count and conditions.

Parameters

agt	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

env	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 cointaining all Agent instances from the model, initial values of the variables and names of the persistent values.

Parameters

lm	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

globalModel	Pointer to a GlobalModel 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 Agant 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 cointaining 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 Agant into a string containing name of the agent, its initial state, transitions with their local and global names, shared count and conditions.

Parameters

agt	Pointer to an Agent 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

env	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 cointaining all Agent instances from the model, initial values of the variables and names of the persistent values.

Parameters

lm	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

globalModel	Pointer to a GlobalModel to print into the console.

6.18 Utils.hpp

Go to the documentation of this file.

```
#ifndef STV_TYPES
#define STV_TYPES

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

using namespace std;

string envToString(map<string, int> env);
string agentToString(Agent* agt);
string localModelsToString(LocalModels* lm);
void outputGlobalModel(GlobalModel* globalModel);
unsigned long getMemCap();

#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

prefix	A prefix string to append to the front of the entry.
h	A pointer to the HistoryEntry 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

prefix	A prefix string to append to the front of every entry.
gs	Pointer to a GlobalState.
st	Enum with a verification status of a global state.
reason	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

status	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

status	Enum value to be converted.

Returns

Verification status converted into a string.

6.21 Verification.hpp

Go to the documentation of this file.

```
#ifndef SELENE_VERIFICATION
#define SELENE_VERIFICATION

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

string verStatusToStr(GlobalStateVerificationStatus status);

enum HistoryEntryType {
    DECISION,
    STATE_STATUS,
    CONTEXT,
    MARK_DECISION_AS_INVALID,
};

struct HistoryEntry {
    HistoryEntryType type;
    GlobalState* globalState;
    GlobalTransition* decision;
    bool globalTransitionControlled;
    GlobalStateVerificationStatus prevStatus;
    GlobalStateVerificationStatus newStatus;
    int depth;
    HistoryEntry* prev;
    HistoryEntry* next;
    string toString() {
        char buff[1024] = { 0 };
        if (this->type == HistoryEntryType::DECISION) {
            snprintf(buff, sizeof(buff), "decision in %s: to %s", this->globalState->hash.c_str(),
this->decision->to->hash.c_str());
        }
        else if (this->type == HistoryEntryType::STATE_STATUS) {
            snprintf(buff, sizeof(buff), "stateVerStatus of %s: %s -> %s",
this->globalState->hash.c_str(), verStatusToStr(this->prevStatus).c_str(),
verStatusToStr(this->newStatus).c_str());
        }
        else if (this->type == HistoryEntryType::CONTEXT) {
            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");
        }
        else if (this->type == HistoryEntryType::MARK_DECISION_AS_INVALID) {
            snprintf(buff, sizeof(buff), "markInvalid in %s: to %s", this->globalState->hash.c_str(),
this->decision->to->hash.c_str());
        }
        return string(buff);
    };
};

class HistoryDbg {
public:
    vector<pair<HistoryEntry*, char» entries;
    HistoryDbg();
    ~HistoryDbg();
    void addEntry(HistoryEntry* entry);
    void markEntry(HistoryEntry* entry, char chr);
    void print(string prefix);
    HistoryEntry* cloneEntry(HistoryEntry* entry);
};

// On-the-fly traversal mode
enum Mode {
    NORMAL,
    REVERT,
    RESTORE,
};

class Verification {
public:
    Verification(GlobalModelGenerator* generator);
    ~Verification();
    bool verify();
protected:
    Mode mode;
    GlobalState* revertToGlobalState;
    stack<HistoryEntry*> historyToRestore;
    GlobalModelGenerator* generator;
    SeleneFormula* seleneFormula;
    HistoryEntry* historyStart;
    HistoryEntry* historyEnd;
    bool verifyLocalStates(set<LocalState*>* localStates);
    bool verifyGlobalState(GlobalState* globalState, int depth);
    bool isGlobalTransitionControlledByCoalition(GlobalTransition* globalTransition);
    bool isAgentInCoalition(Agent* agent);
    EpistemicClass* getEpistemicClassForGlobalState(GlobalState* globalState);
    bool areGlobalStatesInTheSameEpistemicClass(GlobalState* globalState1, GlobalState* globalState2);
    void addHistoryDecision(GlobalState* globalState, GlobalTransition* ecision);
    void addHistoryStateStatus(GlobalState* globalState, GlobalStateVerificationStatus prevStatus,
GlobalStateVerificationStatus newStatus);
    void addHistoryContext(GlobalState* globalState, int depth, GlobalTransition* decision, bool
globalTransitionControlled);
    void addHistoryMarkDecisionAsInvalid(GlobalState* globalState, GlobalTransition* decision);
    HistoryEntry* newHistoryMarkDecisionAsInvalid(GlobalState* globalState, GlobalTransition*
decision);
    bool revertLastDecision(int depth);
    void undoLastHistoryEntry(bool freeMemory);
    void undoHistoryUntil(HistoryEntry* historyEntry, bool inclusive, int depth);
    void printCurrentHistory(int depth);
};

#endif // SELENE_VERIFICATION
```

Index