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

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 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**

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:

- src/Verification.hpp
- src/Verification.cpp

5.30 HistoryEntry Struct Reference

Public Member Functions

- string **toString** ()

Public Attributes

- HistoryEntryType **type**
- GlobalState * **globalState**
- GlobalTransition * **decision**
- bool **globalTransitionControlled**
- GlobalStateVerificationStatus **prevStatus**
- GlobalStateVerificationStatus **newStatus**
- int **depth**
- HistoryEntry * **prev**
- HistoryEntry * **next**

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 * > **varAsssignments**
- 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**
- GlobalState * **revertToGlobalState**
- stack< HistoryEntry * > **historyToRestore**
- GlobalModelGenerator * **generator**
- SeleneFormula * **seleneFormula**
- HistoryEntry * **historyStart**
- HistoryEntry * **historyEnd**

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:

- 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_1v_1cv_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
00014    public:
00015       // metoda do wyliczenia wartości wyrażenia zależna od typu węzła
00016       virtual int eval( Environment& env ) = 0;
00017 };
00018
00019 // węzeł dla stałej
00020 class ExprConst: public ExprNode {
00021
00022    // argumenty
00023    int val;
00024
00025    public:
00026       ExprConst(int _val): val(_val) {};
00027       virtual int eval( Environment& env );
00028 };
00029
00030 // węzeł dla identyfikatora
00031 class ExprIdent: public ExprNode {
00032
00033    // argumenty
00034    string ident;
00035
00036    public:
00037       ExprIdent(string _ident): ident(_ident) {};
00038       virtual int eval( Environment& env );
00039 };
00040
00041 // węzeł dla dodawań
00042 class ExprAdd: public ExprNode {
00043
00044    // argumenty
00045    ExprNode *larg, *rarg;
00046
00047    public:
00048       ExprAdd(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00049       virtual int eval( Environment& env );
00050 };
00051
00052 // węzeł dla odejmowań
00053 class ExprSub: public ExprNode {
00054
00055    // argumenty
00056    ExprNode *larg, *rarg;
00057
00058    public:
00059       ExprSub(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00060       virtual int eval( Environment& env );
00061 };
00062
00063 // węzeł dla mnożeń
00064 class ExprMul: public ExprNode {
00065
00066    // argumenty
00067    ExprNode *larg, *rarg;
00068
00069    public:
00070       ExprMul(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00071       virtual int eval( Environment& env );
00072 };
```

```
00073
00074 // węzeł dla dzieleń
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,
    string _endState, ExprNode *_cond, set<Assignment*> *_assign):
00045             shared(_shared), patternName(_patternName), matchName(_matchName),
00046             startState(_startState), endState(_endState), condition(_cond), assignments(_assign) {};
00047
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 zmienną/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 {
00048     string name;
00049
00051     int initialValue;
00052
00054     bool persistent;
00055
00057     Agent *agent;
00058 };
00059
00061 struct Condition {
00062     Var* var;
00063     ConditionOperator conditionOperator;
00064     int comparedValue;
00065 };
00066
00067 struct Formula {
00068     set<Agent*> coalition;
00069 };
00070
00071 class Agent {
00072     public:
00073         int id;
00074         string name;
00075         set<Var*> vars;
00076         Agent(int _id, string _name):id(_id), name(_name) {};
00077
00078         LocalState* initState;
00079         vector<LocalState*> localStates; // localStates[i].id == i
00080         vector<LocalTransition*> localTransitions; // localTransitions[i].id == i
00081
00082         // sprawdź, czy stan nie został już wygenerowany
00083         LocalState* includesState(LocalState *state);
00084 };
00085
00086 class LocalState {
00087     public:
00088         // Data
00089         int id;
00090         string name;
00091         map<Var*, int> vars;
00092         // alternatywna wersja - może wystarczy
00093         map<string, int> environment;
00094
00095         // komparator
00096         bool compare(LocalState *state);
00097
00098         // Bindings
00099         Agent* agent;
00100         set<LocalTransition*> localTransitions;
00101 };
00102
00103 // to jest zbędne
00104 struct VarAssignment {
00105     Var* dstVar;
00106     VarAssignmentType type; // zbędne
00107     Var* srcVar;
00108     int value;
00109 };
00110
00111 struct LocalTransition {
00112     // Data
00113     int id;
00114     string name;
00115     string localName; // if empty => same as name
00116     bool isShared;
00117     int sharedCount;
00118     set<Condition*> conditions;
00119     set<VarAssignment*> varAsssignments;
00120
00121     // Bindings
00122     Agent* agent;
00123     LocalState* from;
00124     LocalState* to;
00125     set<LocalTransition*> sharedLocalTransitions;
00126 };
00127
00128 struct LocalModels {
00129     vector<Agent*> agents; // agents[i].id == i
00130     map<string, Var*> vars; // vars[str].name == str
00131 };
00132
00133 struct GlobalModel {
00134     // Data
00135     vector<Agent*> agents; // agents[i].id == i
00136     Formula* formula;
00137
```

```
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 Utils.hpp

```
00001 #ifndef STV_TYPES
00002 #define STV_TYPES
00003
00004 #include "Types.hpp"
00005 #include "Constants.hpp"
00006 #include <map>
00007 #include <string>
00008 #include <unistd.h>
00009 #include <sys/time.h>
00010 #include <iostream>
00011 #include <fstream>
00012
00013 using namespace std;
00014
00015 string envToString(map<string, int> env);
00016 string agentToString(Agent* agt);
00017 string localModelsToString(LocalModels* lm);
00018 void outputGlobalModel(GlobalModel* globalModel);
00019 unsigned long getMemCap();
00020
00021 #endif // STV_TYPES
```

6.9 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**

6.9.1 Detailed Description

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

6.9.2 Function Documentation

6.9.2.1 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.

6.9.2.2 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".

6.9.2.3 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.

6.10 Verification.hpp

```
00001 #ifndef SELENE_VERIFICATION
00002 #define SELENE_VERIFICATION
00003
00004 #include <stack>
00005 #include "Types.hpp"
00006 #include "GlobalModelGenerator.hpp"
00007 #include "SeleneFormula.hpp"
00008
00009 string verStatusToStr(GlobalStateVerificationStatus status);
00010
00011 enum HistoryEntryType {
00012     DECISION,
00013     STATE_STATUS,
00014     CONTEXT,
00015     MARK_DECISION_AS_INVALID,
00016 };
00017 struct HistoryEntry {
00018     HistoryEntryType type;
00019     GlobalState* globalState;
00020     GlobalTransition* decision;
00021     bool globalTransitionControlled;
00022     GlobalStateVerificationStatus prevStatus;
00023     GlobalStateVerificationStatus newStatus;
00024     int depth;
00025     HistoryEntry* prev;
00026     HistoryEntry* next;
00027     string toString() {
00028         char buff[1024] = { 0 };
00029         if (this->type == HistoryEntryType::DECISION) {
00030             snprintf(buff, sizeof(buff), "decision in %s: to %s", this->globalState->hash.c_str(),
      this->decision->to->hash.c_str());
00031         }
00032         else if (this->type == HistoryEntryType::STATE_STATUS) {
00033             snprintf(buff, sizeof(buff), "stateVerStatus of %s: %s -> %s",
      this->globalState->hash.c_str(), verStatusToStr(this->prevStatus).c_str(),
      verStatusToStr(this->newStatus).c_str());
00034         }
00035         else if (this->type == HistoryEntryType::CONTEXT) {
```

```
00036             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");
00037         }
00038         else if (this->type == HistoryEntryType::MARK_DECISION_AS_INVALID) {
00039             snprintf(buff, sizeof(buff), "markInvalid in %s: to %s", this->globalState->hash.c_str(),
      this->decision->to->hash.c_str());
00040         }
00041         return string(buff);
00042     };
00043 };
00044
00045 class HistoryDbg {
00046 public:
00047     vector<pair<HistoryEntry*, char» entries;
00048     HistoryDbg();
00049     ~HistoryDbg();
00050     void addEntry(HistoryEntry* entry);
00051     void markEntry(HistoryEntry* entry, char chr);
00052     void print(string prefix);
00053     HistoryEntry* cloneEntry(HistoryEntry* entry);
00054 };
00055
00056 // On-the-fly traversal mode
00057 enum Mode {
00058     NORMAL,
00059     REVERT,
00060     RESTORE,
00061 };
00062
00063 class Verification {
00064 public:
00065     Verification(GlobalModelGenerator* generator);
00066     ~Verification();
00067     bool verify();
00068 protected:
00069     Mode mode;
00070     GlobalState* revertToGlobalState;
00071     stack<HistoryEntry*> historyToRestore;
00072     GlobalModelGenerator* generator;
00073     SeleneFormula* seleneFormula;
00074     HistoryEntry* historyStart;
00075     HistoryEntry* historyEnd;
00076     bool verifyLocalStates(set<LocalState*>* localStates);
00077     bool verifyGlobalState(GlobalState* globalState, int depth);
00078     bool isGlobalTransitionControlledByCoalition(GlobalTransition* globalTransition);
00079     bool isAgentInCoalition(Agent* agent);
00080     EpistemicClass* getEpistemicClassForGlobalState(GlobalState* globalState);
00081     bool areGlobalStatesInTheSameEpistemicClass(GlobalState* globalState1, GlobalState* globalState2);
00082     void addHistoryDecision(GlobalState* globalState, GlobalTransition* ecision);
00083     void addHistoryStateStatus(GlobalState* globalState, GlobalStateVerificationStatus prevStatus,
      GlobalStateVerificationStatus newStatus);
00084     void addHistoryContext(GlobalState* globalState, int depth, GlobalTransition* decision, bool
      globalTransitionControlled);
00085     void addHistoryMarkDecisionAsInvalid(GlobalState* globalState, GlobalTransition* decision);
00086     HistoryEntry* newHistoryMarkDecisionAsInvalid(GlobalState* globalState, GlobalTransition*
      decision);
00087     bool revertLastDecision(int depth);
00088     void undoLastHistoryEntry(bool freeMemory);
00089     void undoHistoryUntil(HistoryEntry* historyEntry, bool inclusive, int depth);
00090     void printCurrentHistory(int depth);
00091 };
00092
00093 #endif // SELENE_VERIFICATION
```

Index