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

Represents a single agent loaded from the description from a file.

```
#include <nodes.hpp>
```

Public Member Functions

- **AgentTemplate** ()

  Constructor for an AgentTemplate.
- virtual AgentTemplate & setIdent (string _ident)

  Set the identifier of an agent.
- virtual AgentTemplate & setInitState (string _startState)

  Set the initial state of the agent.
- virtual AgentTemplate & addLocal (set< string > *variables)

  Adds local variables to an agent.
- virtual AgentTemplate & addPersistent (set< string > *variables)

  Adds persistent variables to an agent.
- virtual AgentTemplate & addInitial (set< Assignment * > *assigns)

  Adds initial assignments.
- virtual AgentTemplate & addTransition (TransitionTemplate *_transition)

  Adds a transition to the agent.
- virtual Agent * generateAgent (int id)

  Generate a new agent for the model.

5.2.1 Detailed Description

Represents a single agent loaded from the description from a file.

5.2.2 Member Function Documentation

5.2.2.1 addInitial()

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

Adds initial assignments.

Sets initial values of agent's variables.

Parameters

assigns	Assignments to be added.

Returns

Returns a pointer to self.

Parameters

assigns	Set of variables to assign.

Returns

Returns itself.

5.2.2.2 addLocal()

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

Adds local variables to an agent.

Adds local variables to the agent.

Parameters

variables	Set of variables to be added.

Returns

Returns a pointer to self.

Parameters

variables	A pointer to a set of strings with the variables to be added.

Returns

Returns itself.

5.2.2.3 addPersistent()

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

Adds persistent variables to an agent.

Adds persistent variables to the agent.

Parameters

variables	Set of variables to be added.

Returns

Returns a pointer to self.

Parameters

variables	A pointer to a set of strings with the variables to be added.

Returns

Returns itself.

5.2.2.4 addTransition()

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

Adds a transition to the agent.

Adds a transition for the agent.

Parameters

_transition	Transition to be added.

Returns

Returns a pointer to self.

Parameters

_transition	Transition to be added.

Returns

Returns itself.

5.2.2.5 generateAgent()

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

Generate a new agent for the model.

Generates an agent for the model.

Parameters

id	Identification number defining a new Agent.

Returns

Returns a pointer to a new Agent.

Parameters

id	Identifier of the new Agent.

Returns

Returns a pointer to a newly created Agent.

5.2.2.6 setIdent()

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

Set the identifier of an agent.

Sets the identifier of an agent.

Parameters

_ident	New agent identifier.

Returns

Returns a pointer to self.

Parameters

_ident	String with a new identifier.

Returns

Returns itself.

5.2.2.7 setInitState()

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

Set the initial state of the agent.

Sets initial state of an agent.

Parameters

_startState	New inital agent state.

Returns

Returns a pointer to self.

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)

  Make an assignment in a given environment.

Public Attributes

- string **ident**

  To what we should assign a value.
- ExprNode * **value**

  A value to be assigned.

5.3.1 Detailed Description

Represents an assingment.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 Assignment()

```
Assignment::Assignment (
            string _ident,
            ExprNode * _exp )  [inline]
```

Constructor for an Assignment class.

Parameters

_ident	To what we should assign a value.
_exp	A value to be assigned.

5.3.3 Member Function Documentation

5.3.3.1 assign()

```
virtual void Assignment::assign (
            Environment & env )  [inline], [virtual]
```

Make an assignment in a given environment.

Parameters

env	Environment in which to make an assignment.

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

- nodes.hpp

5.4 Cfg Struct Reference

Public Attributes

- char * **fname**

- char **stv_mode**
- bool **output_local_models**
- bool **output_global_model**
- 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**

  Transition that was already selected in this epistemic class. Model has to choose this transition if it is already set.

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)
  Addition expression constructor.
- virtual int eval (Environment &env)
  Calculates the expression value.

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

5.7.1 Detailed Description

Node for addition.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 ExprAdd()

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

Addition expression constructor.

Parameters

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

5.7.3 Member Function Documentation

5.7.3.1 eval()

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

Calculates the expression value.

Parameters

env	Environment values.

Returns

Returns an integer.

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)
  Logic AND expression constructor.
- virtual int eval (Environment &env)
  Calculates the expression value.

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

5.8.1 Detailed Description

Node for AND operator.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 ExprAnd()

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

Logic AND expression constructor.

Parameters

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

5.8.3 Member Function Documentation

5.8.3.1 eval()

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

Calculates the expression value.

Parameters

env	Environment values.

Returns

Returns an integer.

Implements ExprNode.

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

- expressions.hpp
- expressions.cc

5.9 ExprConst Class Reference

Node for a constant.

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

Public Member Functions

- ExprConst (int _val)

  Constant expression constructor.
- virtual int eval (Environment &env)

  Calculates the expression value.

- virtual int eval (Environment &env)=0

  Calculates the expression value.

5.9.1 Detailed Description

Node for a constant.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 ExprConst()

```
ExprConst::ExprConst (
            int _val )  [inline]
```

Constant expression constructor.

Parameters

_val	ExprConst value.

5.9.3 Member Function Documentation

5.9.3.1 eval()

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

Calculates the expression value.

Parameters

env	Environment values.

Returns

Returns an integer.

Implements ExprNode.

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

- expressions.hpp
- expressions.cc

5.10 ExprDiv Class Reference

Node for division.

`#include <expressions.hpp>`

Public Member Functions

- ExprDiv (ExprNode *_larg, ExprNode *_rarg)
  Division expression constructor.
- virtual int eval (Environment &env)
  Calculates the expression value.

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

5.10.1 Detailed Description

Node for division.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 ExprDiv()

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

Division expression constructor.

Parameters

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

5.10.3 Member Function Documentation

5.10.3.1 eval()

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

Calculates the expression value.

Parameters

env	Environment values.

Returns

Returns an integer.

Implements ExprNode.

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

- expressions.hpp
- expressions.cc

5.11 ExprEq Class Reference

Node for "==" operator.

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

Public Member Functions

- ExprEq (ExprNode *_larg, ExprNode *_rarg)
  Equals expression constructor.
- virtual int eval (Environment &env)
  Calculates the expression value.
- virtual int eval (Environment &env)=0
  Calculates the expression value.

5.11.1 Detailed Description

Node for "==" operator.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 ExprEq()

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

Equals expression constructor.

Parameters

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

5.11.3 Member Function Documentation

5.11.3.1 eval()

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

Calculates the expression value.

Parameters

env	Environment values.

Returns

Returns an integer.

Implements ExprNode.

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

- expressions.hpp
- expressions.cc

5.12 ExprGe Class Reference

Node for ">=" operator.

`#include <expressions.hpp>`

Public Member Functions

- ExprGe (ExprNode *_larg, ExprNode *_rarg)
  Greater or equal expression constructor.
- virtual int eval (Environment &env)
  Calculates the expression value.

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

5.12.1 Detailed Description

Node for ">=" operator.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 ExprGe()

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

Greater or equal expression constructor.

Parameters

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

5.12.3 Member Function Documentation

5.12.3.1 eval()

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

Calculates the expression value.

Parameters

env	Environment values.

Returns

Returns an integer.

Implements ExprNode.

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

- expressions.hpp
- expressions.cc

5.13 ExprGt Class Reference

Node for ">" operator.

`#include <expressions.hpp>`

Public Member Functions

- ExprGt (ExprNode *_larg, ExprNode *_rarg)
  Greater than expression constructor.
- virtual int eval (Environment &env)
  Calculates the expression value.

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

5.13.1 Detailed Description

Node for ">" operator.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 ExprGt()

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

Greater than expression constructor.

Parameters

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

5.13.3 Member Function Documentation

5.13.3.1 eval()

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

Calculates the expression value.

Parameters

env	Environment values.

Returns

Returns an integer.

Implements ExprNode.

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

- expressions.hpp
- expressions.cc

5.14 ExprIdent Class Reference

Node for an identifier.

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

Public Member Functions

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

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

5.14.1 Detailed Description

Node for an identifier.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 ExprIdent()

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

Identifier expression constructor.

Parameters

_ident	ExprIdent value.

5.14.3 Member Function Documentation

5.14.3.1 eval()

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

Calculates the expression value.

Parameters

env	Environment values.

Returns

Returns an integer.

Parameters

env	

Returns

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)
  Less or equal expression constructor.
- virtual int eval (Environment &env)
  Calculates the expression value.

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

5.15.1 Detailed Description

Node for "<=" operator.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 ExprLe()

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

Less or equal expression constructor.

Parameters

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

5.15.3 Member Function Documentation

5.15.3.1 eval()

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

Calculates the expression value.

Parameters

env	Environment values.

Returns

Returns an integer.

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)
  Less than expression constructor.
- virtual int eval (Environment &env)
  Calculates the expression value.
- virtual int eval (Environment &env)=0
  Calculates the expression value.

5.16.1 Detailed Description

Node for "<" operator.

5.16.2 Constructor & Destructor Documentation

5.16.2.1 ExprLt()

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

Less than expression constructor.

Parameters

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

5.16.3 Member Function Documentation

5.16.3.1 eval()

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

Calculates the expression value.

Parameters

env	Environment values.

Returns

Returns an integer.

Implements ExprNode.

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

- expressions.hpp
- expressions.cc

5.17 ExprMul Class Reference

Node for multiplication.

`#include <expressions.hpp>`

Public Member Functions

- ExprMul (ExprNode *_larg, ExprNode *_rarg)
  Multiplication expression constructor.
- virtual int eval (Environment &env)
  Calculates the expression value.

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

5.17.1 Detailed Description

Node for multiplication.

5.17.2 Constructor & Destructor Documentation

5.17.2.1 ExprMul()

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

Multiplication expression constructor.

Parameters

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

5.17.3 Member Function Documentation

5.17.3.1 eval()

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

Calculates the expression value.

Parameters

env	Environment values.

Returns

Returns an integer.

Implements ExprNode.

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

- expressions.hpp
- expressions.cc

5.18 ExprNe Class Reference

Node for "!=" operator.

`#include <expressions.hpp>`

Public Member Functions

- ExprNe (ExprNode *_larg, ExprNode *_rarg)
  Not equals expression constructor.
- virtual int eval (Environment &env)
  Calculates the expression value.
- virtual int eval (Environment &env)=0
  Calculates the expression value.

5.18.1 Detailed Description

Node for "!=" operator.

5.18.2 Constructor & Destructor Documentation

5.18.2.1 ExprNe()

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

Not equals expression constructor.

Parameters

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

5.18.3 Member Function Documentation

5.18.3.1 eval()

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

Calculates the expression value.

Parameters

env	Environment values.

Returns

Returns an integer.

Implements ExprNode.

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

- expressions.hpp
- expressions.cc

5.19 ExprNode Class Reference

Base node for expressions.

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

Public Member Functions

- virtual int eval (Environment &env)=0

  Calculates the expression value.

5.19.1 Detailed Description

Base node for expressions.

5.19.2 Member Function Documentation

5.19.2.1 eval()

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

Calculates the expression value.

Parameters

env	Environment values.

Returns

Returns an integer.

Implemented in ExprConst, ExprIdent, ExprAdd, ExprSub, ExprMul, ExprDiv, ExprRem, ExprAnd, ExprOr, ExprNot, ExprEq, ExprNe, ExprLt, ExprLe, ExprGt, and ExprGe.

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)
  Logic NOT expression constructor.
- virtual int eval (Environment &env)
  Calculates the expression value.

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

5.20.1 Detailed Description

Node for NOT operator.

5.20.2 Constructor & Destructor Documentation

5.20.2.1 ExprNot()

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

Logic NOT expression constructor.

Parameters

_arg	Calculates the expression value.

5.20.3 Member Function Documentation

5.20.3.1 eval()

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

Calculates the expression value.

Parameters

env	Environment values.

Returns

Returns an integer.

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)
  Logic OR expression constructor.
- virtual int eval (Environment &env)
  Calculates the expression value.

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

5.21.1 Detailed Description

Node for OR operator.

5.21.2 Constructor & Destructor Documentation

5.21.2.1 ExprOr()

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

Logic OR expression constructor.

Parameters

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

5.21.3 Member Function Documentation

5.21.3.1 eval()

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

Calculates the expression value.

Parameters

env	Environment values.

Returns

Returns an integer.

Implements ExprNode.

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

- expressions.hpp
- expressions.cc

5.22 ExprRem Class Reference

Node for modulo.

`#include <expressions.hpp>`

Public Member Functions

- ExprRem (ExprNode *_larg, ExprNode *_rarg)
  Modulo expression constructor.
- virtual int eval (Environment &env)
  Calculates the expression value.

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

5.22.1 Detailed Description

Node for modulo.

5.22.2 Constructor & Destructor Documentation

5.22.2.1 ExprRem()

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

Modulo expression constructor.

Parameters

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

5.22.3 Member Function Documentation

5.22.3.1 eval()

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

Calculates the expression value.

Parameters

env	Environment values.

Returns

Returns an integer.

Implements ExprNode.

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

- expressions.hpp
- expressions.cc

5.23 ExprSub Class Reference

Node for subtraction.

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

Public Member Functions

- ExprSub (ExprNode *_larg, ExprNode *_rarg)
  Subtraction expression constructor.
- virtual int eval (Environment &env)
  Calculates the expression value.
- virtual int eval (Environment &env)=0
  Calculates the expression value.

5.23.1 Detailed Description

Node for subtraction.

5.23.2 Constructor & Destructor Documentation

5.23.2.1 ExprSub()

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

Subtraction expression constructor.

Parameters

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

5.23.3 Member Function Documentation

5.23.3.1 eval()

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

Calculates the expression value.

Parameters

env	Environment values.

Returns

Returns an integer.

Implements ExprNode.

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

- expressions.hpp
- expressions.cc

5.24 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

Stores the local models, formula and a global model.

`#include <GlobalModelGenerator.hpp>`

Public Member Functions

- **GlobalModelGenerator** ()

  Constructor for GlobalModelGenerator class.
- ∼**GlobalModelGenerator** ()

  Destructor for GlobalModelGenerator class.
- GlobalState ∗ initModel (LocalModels ∗localModels, Formula ∗formula)

  Initializes a global model from local models and a formula.
- void expandState (GlobalState ∗state)

  Goes through all GlobalTransition in a given GlobalState and creates new GlobalStates connected to the given one.
- void **expandAllStates** ()

  Expands the states starting from the initial GlobalState and continues until there are no more states to expand.
- GlobalModel ∗ getCurrentGlobalModel ()

  Get for a GlobalModel used in initialization.
- Formula ∗ getFormula ()

  Get for the Formula used in initialization.

Protected Member Functions

- GlobalState ∗ generateInitState ()

  Generates initial state of the model from GlobalModel in memory.
- GlobalState ∗ generateStateFromLocalStates (set< LocalState ∗ > ∗localStates, set< LocalTransition ∗ > ∗viaLocalTransitions, GlobalState ∗prevGlobalState)

  Creates a new GlobalState using some of the internally known model data and given local states, transitions that were uset to get there and the previous global state.
- void generateGlobalTransitions (GlobalState ∗fromGlobalState, set< LocalTransition ∗ > localTransitions, map< Agent ∗, vector< LocalTransition ∗ > > transitionsByAgent)

  Adds all shared global transitions to a GlobalState.
- bool checkLocalTransitionConditions (LocalTransition ∗localTransition, GlobalState ∗globalState)

  Checks if all conditions for a given local transition in a given global state are fulfilled.
- string computeEpistemicClassHash (set< LocalState ∗ > ∗localStates, Agent ∗agent)

  Creates a hash from a set of LocalState and an Agent.
- string computeGlobalStateHash (set< LocalState ∗ > ∗localStates)

  Creates a hash from a set of LocalState.
- EpistemicClass ∗ findOrCreateEpistemicClass (set< LocalState ∗ > ∗localStates, Agent ∗agent)

  Checks if a set of LocalState is already an epistemic class for a given Agent, if not, creates a new one.
- GlobalState ∗ findGlobalStateInEpistemicClass (set< LocalState ∗ > ∗localStates, EpistemicClass ∗epistemicClass)

  Gets a GlobalState from an EpistemicClass if it exists in that episcemic class.

Protected Attributes

- LocalModels * **localModels**

  LocalModels used in initModel.
- Formula * **formula**

  Formula used in initModel.
- GlobalModel * **globalModel**

  GlobalModel created in initModel.

5.26.1 Detailed Description

Stores the local models, formula and a global model.

5.26.2 Member Function Documentation

5.26.2.1 checkLocalTransitionConditions()

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

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

Parameters

localTransition	Local transition to traverse.
globalState	Current global state.

Returns

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

5.26.2.2 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.

5.26.2.3 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.

5.26.2.4 expandState()

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

Goes through all GlobalTransition in a given GlobalState and creates new GlobalStates connected to the given one.

Parameters

state	A state from which the expansion should start.

5.26.2.5 findGlobalStateInEpistemicClass()

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

Gets a GlobalState from an EpistemicClass if it exists in that episcemic class.

Parameters

localStates	Pointer to a set of pointers to LocalState, from which will be generated a global state hash.
epistemicClass	Epistemic class in which to check if a GlobalState exists.

Returns

Returns a pointer to a GlobalState if it exists in given epistemic class, otherwise returns nullptr.

5.26.2.6 findOrCreateEpistemicClass()

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

Checks if a set of LocalState is already an epistemic class for a given Agent, if not, creates a new one.

Parameters

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.

5.26.2.7 generateGlobalTransitions()

```
void GlobalModelGenerator::generateGlobalTransitions (
            GlobalState * fromGlobalState,
            set< LocalTransition * > localTransitions,
            map< Agent *, vector< LocalTransition * > > transitionsByAgent )  [protected]
```

Adds all shared global transitions to a GlobalState.

Parameters

fromGlobalState	Global state to add transitions to.
localTransitions	Initially empty, avaliable local transitions by each agent from transitionsByAgent.
transitionsByAgent	Mapped transitions to an agent, only with transitions avaliable for the agent at this moment.

5.26.2.8 generateInitState()

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

Generates initial state of the model from GlobalModel in memory.

Returns

Returns a pointer to an initial GlobalState.

5.26.2.9 generateStateFromLocalStates()

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

Creates a new GlobalState using some of the internally known model data and given local states, transitions that were uset to get there and the previous global state.

Parameters

localStates	LocalStates from which the new GlobalState will be built.
viaLocalTransitions	Pointer to a set of pointers to LocalTransition from which the changes in variables, as a result of traversing through the transition, will be made in a new GlobalState.
prevGlobalState	Pointer to GlobalState from which all persistent variables will be copied over from to the new GlobalState.

Returns

Returns a pointer to a new or already existing in the same epistemic class GlobalModel.

5.26.2.10 getCurrentGlobalModel()

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

Get for a GlobalModel used in initialization.

Returns

Returns a pointer to a global model.

5.26.2.11 getFormula()

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

Get for the Formula used in initialization.

Returns

Returns a pointer to the formula structure.

5.26.2.12 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**

  If false, the state can be still expanded, potentially creating new states, otherwise the expansion of the state already occured and is not necessary.
- GlobalStateVerificationStatus **verificationStatus**

  Current verifivation status of this state.
- set< GlobalTransition * > **globalTransitions**

  Every GlobalTransition in the model.
- set< LocalState * > **localStates**

  Local states of each agent that define this global state.

5.27.1 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**

  Marks if the transition is invalid, true if there is no point in traversing that transition, otherwise false.
- GlobalState * **from**

  Binding to a GlobalState from which this transition goes from.
- GlobalState * **to**

  Binding to a GlobalState from which this transition goes to.
- set< LocalTransition * > **localTransitions**

  Local transitions that define this global transition. A single transition or more in case of shared transitions.

5.28.1 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

Stores history and allows displaying it to the console.

`#include <Verification.hpp>`

Public Member Functions

- **HistoryDbg** ()

  A constructor for HistoryDbg.
- ~**HistoryDbg** ()

  A destructor for HistoryDbg.
- void addEntry (HistoryEntry *entry)

  Adds a HistoryEntry to the debug history.
- void markEntry (HistoryEntry *entry, char chr)

  Marks an entry in the 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 Detailed Description

Stores history and allows displaying it to the console.

5.29.2 Member Function Documentation

5.29.2.1 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.

5.29.2.2 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.

5.29.2.3 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.

5.29.2.4 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**
  Local variables and their values.
- map< string, int > **environment**
  Local variables as a name and their current values.
- Agent * **agent**
  Binding to an Agent.
- set< LocalTransition * > **localTransitions**
  Binding to the set of LocalTransition.

5.32.1 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

A template for the local state.

`#include <nodes.hpp>`

Public Attributes

- string **name**

  Name of the local state.
- set< TransitionTemplate * > **transitions**

  Local transitions going out from this state.

5.33.1 Detailed Description

A template for the local state.

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**

  Conditions that have to be fulfilled for the transition to be avaliable.
- set< VarAssignment * > **varAsssignments**

  Values to be set as a result of the traversal.
- Agent * **agent**

  Binding to an Agent.
- LocalState * **from**

  Binding to a LocalState from which this transition goes from.
- LocalState * **to**

  Binding to a LocalState from which this transition goes to.
- set< LocalTransition * > **sharedLocalTransitions**

  Stores shared transitions from different models.

5.34.1 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

A parser for converting a text file into a model.

```
#include <TestParser.hpp>
```

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 Detailed Description

A parser for converting a text file into a model.

5.37.2 Member Function Documentation

5.37.2.1 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

Represents a meta-transition.

```
#include <nodes.hpp>
```

Public Member Functions

- TransitionTemplate (int _shared, string _patternName, string _matchName, string _startState, string _endState, ExprNode *_cond, set< Assignment * > *_assign)

  TransitionTemplate constructor.

Public Attributes

- int **shared**

  Needed amound of needed agents. -1 if not shared.
- string **patternName**

  Name of the pattern.
- string **matchName**

  Global name for shared transitions.
- string **startState**

  Start state name.
- string **endState**

  End state name.
- ExprNode * **condition**

  Condition expression that has do be fulfilled in that transition.
- set< Assignment * > * **assignments**

  Set of assignments.

5.38.1 Detailed Description

Represents a meta-transition.

5.38.2 Constructor & Destructor Documentation

5.38.2.1 TransitionTemplate()

```
TransitionTemplate::TransitionTemplate (
            int _shared,
            string _patternName,
            string _matchName,
            string _startState,
            string _endState,
            ExprNode * _cond,
            set< Assignment * > * _assign )  [inline]
```

TransitionTemplate constructor.

Parameters

_shared	Needed amound of needed agents. -1 if not shared.
_patternName	Name of the pattern.
_matchName	Global name for shared transitions.
_startState	Start state name.
_endState	End state name.
_cond	Condition expression that has do be fulfilled in that transition.
_assign	Set of assignments.

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

- nodes.hpp

5.39 Var Struct Reference

Represents a variable in the model, containing name, initial value and persistence.

`#include <Types.hpp>`

Public Attributes

- string **name**

  Variable name.
- int **initialValue**

  Initial value of the variable.
- bool **persistent**

  True if variable is persistent, i.e. it should appear in all states in the model, false otherwise.
- Agent * **agent**

  Reference to an agent, to which this variable belongs to.

5.39.1 Detailed Description

Represents a variable in the model, containing name, initial value and persistence.

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

- Types.hpp

5.40 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

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

`#include <Verification.hpp>`

Public Member Functions

- Verification (GlobalModelGenerator *generator)

  Constructor for Verification.
- ~**Verification** ()

  Destructor for Verification.
- bool verify ()

  Starts the process of formula verification on a model.

Protected Member Functions

- bool verifyLocalStates (set< LocalState * > *localStates)

  Verifies a set of LocalState that a GlobalState is composed of with a hardcoded formula.
- bool verifyGlobalState (GlobalState *globalState, int depth)

  Recursively verifies GlobalState.
- bool isGlobalTransitionControlledByCoalition (GlobalTransition *globalTransition)

  Checks if any of the LocalTransition in a given GlobalTransition has an Agent in a coalition in the formula.
- bool isAgentInCoalition (Agent *agent)

  Checks if the Agent is in a coalition based on the formula in a GlobalModelGenerator.
- EpistemicClass * getEpistemicClassForGlobalState (GlobalState *globalState)

  Gets the EpistemicClass for the agent in passed GlobalState, i.e. transitions from indistinguishable state from certain other states for an agent to other states.
- bool areGlobalStatesInTheSameEpistemicClass (GlobalState *globalState1, GlobalState *globalState2)

  Compares two GlobalState and checks if their EpistemicClass is the same.
- void addHistoryDecision (GlobalState *globalState, GlobalTransition *ecision)

  Creates a HistoryEntry of the type DECISION and puts it on top of the stack of the decision history.
- void addHistoryStateStatus (GlobalState *globalState, GlobalStateVerificationStatus prevStatus, GlobalStateVerificationStatus newStatus)

  Creates a HistoryEntry of the type STATE_STATUS and puts it to the top of the decision history.
- void addHistoryContext (GlobalState *globalState, int depth, GlobalTransition *decision, bool globalTransitionControlled)

  Creates a HistoryEntry of the type CONTEXT and puts it to the top of the decision history.
- void addHistoryMarkDecisionAsInvalid (GlobalState *globalState, GlobalTransition *decision)

  Creates a HistoryEntry of the type MARK_DECISION_AS_INVALID and puts it to the top of the decision history.
- HistoryEntry * newHistoryMarkDecisionAsInvalid (GlobalState *globalState, GlobalTransition *decision)

  Creates a HistoryEntry of the type MARK_DECISION_AS_INVALID and returns it.
- bool revertLastDecision (int depth)

  Reverts GlobalState and history to the previous decision state.
- void undoLastHistoryEntry (bool freeMemory)

  Removes the top entry of the history stack.
- void undoHistoryUntil (HistoryEntry *historyEntry, bool inclusive, int depth)

  Rolls back the history entries up to the certain HistoryEntry.
- void printCurrentHistory (int depth)

  Prints current history to the console.

Protected Attributes

- Mode **mode**

  Current mode of model traversal.
- GlobalState * **revertToGlobalState**

  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 Detailed Description

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

5.41.2 Constructor & Destructor Documentation

5.41.2.1 Verification()

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

Constructor for Verification.

Parameters

generator	Pointer to GlobalModelGenerator

5.41.3 Member Function Documentation

5.41.3.1 addHistoryContext()

```
void Verification::addHistoryContext (
            GlobalState * globalState,
            int depth,
```

```
            GlobalTransition * decision,
            bool globalTransitionControlled )  [protected]
```

Creates a HistoryEntry of the type CONTEXT and puts it to the top of the decision history.

Parameters

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.

5.41.3.2 addHistoryDecision()

```
void Verification::addHistoryDecision (
            GlobalState * globalState,
            GlobalTransition * decision )  [protected]
```

Creates a HistoryEntry of the type DECISION and puts it on top of the stack of the decision history.

Parameters

globalState	Pointer to a GlobalState of the model.
decision	Pointer to a GlobalTransition that is to be recorded in the decision history.

5.41.3.3 addHistoryMarkDecisionAsInvalid()

```
void Verification::addHistoryMarkDecisionAsInvalid (
            GlobalState * globalState,
            GlobalTransition * decision )  [protected]
```

Creates a HistoryEntry of the type MARK_DECISION_AS_INVALID and puts it to the top of the decision history.

Parameters

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

5.41.3.4 addHistoryStateStatus()

```
void Verification::addHistoryStateStatus (
            GlobalState * globalState,
```

```
            GlobalStateVerificationStatus prevStatus,
            GlobalStateVerificationStatus newStatus )  [protected]
```

Creates a HistoryEntry of the type STATE_STATUS and puts it to the top of the decision history.

Parameters

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

5.41.3.5 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.

5.41.3.6 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.

5.41.3.7 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.

5.41.3.8 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.

5.41.3.9 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.

5.41.3.10 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.

5.41.3.11 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.

5.41.3.12 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.

5.41.3.13 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.

5.41.3.14 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.

5.41.3.15 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.

5.41.3.16 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

```
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.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

  Stores the local models, formula and a global model.

6.3.1 Detailed Description

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

6.4 GlobalModelGenerator.hpp

Go to the documentation of this file.

```
00001
00007 #ifndef SELENE_GLOBAL_MODEL_GENERATOR
00008 #define SELENE_GLOBAL_MODEL_GENERATOR
00009
00010 #include "Constants.hpp"
00011 #include "Types.hpp"
00012
00013 using namespace std;
00014
00016 class GlobalModelGenerator {
00017 public:
00018     GlobalModelGenerator();
00019     ~GlobalModelGenerator();
00020     GlobalState* initModel(LocalModels* localModels, Formula* formula);
00021     void expandState(GlobalState* state);
00022     void expandAllStates();
00023     GlobalModel* getCurrentGlobalModel();
00024     Formula* getFormula();
00025
00026 protected:
00028     LocalModels* localModels;
00030     Formula* formula;
00032     GlobalModel* globalModel;
00033     GlobalState* generateInitState();
00034     GlobalState* generateStateFromLocalStates(set<LocalState*>* localStates, set<LocalTransition*>*
      viaLocalTransitions, GlobalState* prevGlobalState);
00035     void generateGlobalTransitions(GlobalState* fromGlobalState, set<LocalTransition*>
      localTransitions, map<Agent*, vector<LocalTransition*>* transitionsByAgent);
00036     bool checkLocalTransitionConditions(LocalTransition* localTransition, GlobalState* globalState);
00037     string computeEpistemicClassHash(set<LocalState*>* localStates, Agent* agent);
00038     string computeGlobalStateHash(set<LocalState*>* localStates);
00039     EpistemicClass* findOrCreateEpistemicClass(set<LocalState*>* localStates, Agent* agent);
00040     GlobalState* findGlobalStateInEpistemicClass(set<LocalState*>* localStates, EpistemicClass*
      epistemicClass);
00041 };
00042
00043 #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**
  Variable names with their values.

6.6.1 Detailed Description

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

6.7 expressions.hpp

Go to the documentation of this file.

```
00001
00007 #ifndef __EXPRESSIONS_H
00008 #define __EXPRESSIONS_H
00009
00010 #include <string>
00011 #include <map>
00012
00013 using namespace std;
00014
00016 typedef map<string, int> Environment;
00017
00018 // węzeł bazowy dla wyrażeń
00019
00021 class ExprNode {
00022
00023    public:
00024       // metoda do wyliczenia wartości wyrażenia zależna od typu węzła
00025
00029       virtual int eval( Environment& env ) = 0;
00030 };
00031
00032 // węzeł dla stałej
00033
00035 class ExprConst: public ExprNode {
00036
00037    // argumenty
00038
00040    int val;
00041
00042    public:
00045       ExprConst(int _val): val(_val) {};
00046
00050       virtual int eval( Environment& env );
00051 };
00052
00053 // węzeł dla identyfikatora
00054
00056 class ExprIdent: public ExprNode {
00057
00058    // argumenty
00059
00061    string ident;
00062
00063    public:
00066       ExprIdent(string _ident): ident(_ident) {};
00067
00071       virtual int eval( Environment& env );
00072 };
00073
00074 // węzeł dla dodawań
00075
00077 class ExprAdd: public ExprNode {
00078
00079    // argumenty
00080
00082    ExprNode *larg, *rarg;
00083
00084    public:
00088       ExprAdd(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00089
00093       virtual int eval( Environment& env );
00094 };
00095
00096 // węzeł dla odejmowań
00097
00099 class ExprSub: public ExprNode {
00100
00101    // argumenty
00102
00104    ExprNode *larg, *rarg;
00105
00106    public:
```

```
00110       ExprSub(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00111
00115       virtual int eval( Environment& env );
00116 };
00117
00118 // węzeł dla mnożeń
00119
00121 class ExprMul: public ExprNode {
00122
00123    // argumenty
00124
00126    ExprNode *larg, *rarg;
00127
00128    public:
00132       ExprMul(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00133
00137       virtual int eval( Environment& env );
00138 };
00139
00140 // węzeł dla dzieleń
00141
00143 class ExprDiv: public ExprNode {
00144
00145    // argumenty
00146
00148    ExprNode *larg, *rarg;
00149
00150    public:
00154       ExprDiv(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00155
00159       virtual int eval( Environment& env );
00160 };
00161
00162 // węzeł dla reszty z dzielenia
00163
00165 class ExprRem: public ExprNode {
00166
00167    // argumenty
00168
00170    ExprNode *larg, *rarg;
00171
00172    public:
00176       ExprRem(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00177
00181       virtual int eval( Environment& env );
00182 };
00183
00184 // węzeł dla operatora AND
00185
00187 class ExprAnd: public ExprNode {
00188
00189    // argumenty
00190
00192    ExprNode *larg, *rarg;
00193
00194    public:
00198       ExprAnd(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00199
00203       virtual int eval( Environment& env );
00204 };
00205
00206 // węzeł dla operatora OR
00207
00209 class ExprOr: public ExprNode {
00210
00211    // argumenty
00212
00214    ExprNode *larg, *rarg;
00215
00216    public:
00220       ExprOr(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00221
00225       virtual int eval( Environment& env );
00226 };
00227
00228 // węzeł dla operatora NOT
00229
00231 class ExprNot: public ExprNode {
00232
00233    // argumenty
00234
00236    ExprNode *arg;
00237
00238    public:
00241       ExprNot(ExprNode *_arg): arg(_arg) {};
00242
00246       virtual int eval( Environment& env );
```

```
00247 };
00248
00249 // węzeł dla operatora "=="
00250
00252 class ExprEq: public ExprNode {
00253
00254    // argumenty
00255
00257    ExprNode *larg, *rarg;
00258
00259    public:
00263       ExprEq(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00264
00268       virtual int eval( Environment& env );
00269 };
00270
00271 // węzeł dla operatora "!="
00272
00274 class ExprNe: public ExprNode {
00275
00276    // argumenty
00277
00279    ExprNode *larg, *rarg;
00280
00281    public:
00285       ExprNe(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00286
00290       virtual int eval( Environment& env );
00291 };
00292
00293 // węzeł dla operatora "<"
00294
00296 class ExprLt: public ExprNode {
00297
00298    // argumenty
00299
00301    ExprNode *larg, *rarg;
00302
00303    public:
00307       ExprLt(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00308
00312       virtual int eval( Environment& env );
00313 };
00314
00315 // węzeł dla operatora "<="
00316
00318 class ExprLe: public ExprNode {
00319
00320    // argumenty
00321
00323    ExprNode *larg, *rarg;
00324
00325    public:
00329       ExprLe(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00330
00334       virtual int eval( Environment& env );
00335 };
00336
00337 // węzeł dla operatora ">"
00338
00340 class ExprGt: public ExprNode {
00341
00342    // argumenty
00343
00345    ExprNode *larg, *rarg;
00346
00347    public:
00351       ExprGt(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00352
00356       virtual int eval( Environment& env );
00357 };
00358
00359 // węzeł dla operatora ">="
00360
00362 class ExprGe: public ExprNode {
00363
00364    // argumenty
00365
00367    ExprNode *larg, *rarg;
00368
00369    public:
00373       ExprGe(ExprNode *_larg, ExprNode *_rarg): larg(_larg), rarg(_rarg) {};
00374
00378       virtual int eval( Environment& env );
00379 };
00380
00381
```

```
00382 #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

  Represents a meta-transition.
- class LocalStateTemplate

  A template for the local state.
- class AgentTemplate

  Represents a single agent loaded from the description from a file.

6.9.1 Detailed Description

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

6.10 nodes.hpp

Go to the documentation of this file.

```
00001
00007 #ifndef __NODES_H
00008 #define __NODES_H
00009
00010 #include <string>
00011 #include <set>
00012 #include <map>
00013 #include "expressions.hpp"
00014
00015 #include "../Types.hpp"
00016
00017 using namespace std;
00018
00019 /* Klasa reprezentująca przypisanie */
00020
00022 class Assignment {
00023    public:
00025       string ident;
00026       // do czego przypisujemy
00027
00029       ExprNode *value;
00030       // co przypisujemy
00031
00035       Assignment(string _ident, ExprNode *_exp): ident(_ident), value(_exp) {};
00036
00037       // wykonaj przypisanie w danym środowisku
00038
00041       virtual void assign(Environment &env) {
00042          env[ident]=value->eval(env);
00043       };
00044 };
00045
00046 /* Klasa reprezentująca meta-tranzycję */
00047
00049 class TransitionTemplate {
00050    public:
00051       // jeśli -1 to nie ma dzielenia, w p.p. wartość określa łączną liczbę wymaganych agentów
00052
00054       int shared;
00055
00056       // nazwa wzorca
00057
00059       string patternName;
00060
00061       // nazwa do wyszukiwania dla shared
00062
00064       string matchName;
00065
00066       // nazwa stanu początkowego i końcowego
00067
00069       string startState;
00070
00072       string endState;
00073
00074       // wyrażenie warunkowe
00075
00077       ExprNode *condition;
00078
00079       // lista przypisań wartości
00080
00082       set<Assignment*> *assignments;
00083
00092       TransitionTemplate(int _shared, string _patternName, string _matchName, string _startState,
      string _endState, ExprNode *_cond, set<Assignment*> *_assign):
00093             shared(_shared), patternName(_patternName), matchName(_matchName),
00094             startState(_startState), endState(_endState), condition(_cond), assignments(_assign) {};
00095
00096 };
00097
00099 class LocalStateTemplate {
00100    public:
00102       string name;
00103
00105       set<TransitionTemplate*> transitions;
00106 };
00107
00108 /* Klasa reprezentująca pojedynczego agenta po wczytaniu jego opisu z pliku */
00109
00111 class AgentTemplate {
00112       // identyfikator agenta
00113
00115       string ident;
```

```
00116
00117       // stan startowy
00118
00120       string initState;
00121
00122       // zbiór zmiennych lokalnych (local)
00123
00125       set<string>* localVars;
00126
00127       // zbiór zmiennych trwałych (persistent)
00128
00130       set<string>* persistentVars;
00131
00132       // początkowa inicjacja
00133
00135       set<Assignment*>* initialAssignments;
00136
00137       // zbiór tranzycji
00138
00140       set<TransitionTemplate*>* transitions;
00141
00142       // mapa stanów lokalnych potrzebna do wygenerowania modelu
00143
00145       map<string,LocalStateTemplate*> localStateTemplates;
00146
00147       // metoda wyznaczająca węzeł kolejny do danego, zależnie od tranzycji
00148
00149       virtual LocalState* genNextState(LocalState *state, TransitionTemplate *trans);
00150
00151    public:
00152       AgentTemplate();
00153
00154       // ustaw identyfikator agenta
00155
00159       virtual AgentTemplate& setIdent(string _ident);
00160
00161       // ustaw identyfikator agenta
00162
00166       virtual AgentTemplate& setInitState(string _startState);
00167
00168       // dodaj zmienną/zmienne lokalne
00169
00173       virtual AgentTemplate& addLocal(set<string> *variables);
00174
00175       // dodaj zmienne trwałe
00176
00180       virtual AgentTemplate& addPersistent(set<string> *variables);
00181
00182       // dodaj początkowe inicjacje
00183
00187       virtual AgentTemplate& addInitial(set<Assignment*> *assigns);
00188
00189       // dodaj tranzycję
00190
00194       virtual AgentTemplate& addTransition(TransitionTemplate *_transition);
00195
00196       // wygeneruj agenta do modelu
00197
00201       virtual Agent* generateAgent(int id) ;
00202 };
00203
00204 #endif
```

6.11 SeleneFormula.hpp

```
00001 #ifndef SELENE_SELENE_FORMULA
00002 #define SELENE_SELENE_FORMULA
00003
00004 #include "Types.hpp"
00005
00006
00007
00008
00009
00010 class SeleneFormula {
00011 public:
00012     SeleneFormula();
00013     ~SeleneFormula();
00014     virtual bool verifyLocalStates(set<LocalState*>* localStates) = 0;
00015     LocalState* getLocalStateForAgent(string agentName, set<LocalState*>* localStates);
00016     int getLocalStateVar(string varName, LocalState* localState);
00017     inline bool implication(bool left, bool right);
00018 };
```

```
00019
00020
00021
00022
00023
00024 class SeleneFormula1 : public SeleneFormula {
00025 public:
00026     SeleneFormula1();
00027     ~SeleneFormula1();
00028     bool verifyLocalStates(set<LocalState*>* localStates);
00029 protected:
00030 };
00031
00032
00033
00034
00035
00036 #endif // SELENE_SELENE_FORMULA
```

6.12 TestParser.hpp

```
00001
00007 #ifndef __TESTPARSER_HPP
00008 #define __TESTPARSER_HPP
00009
00010 #include "Types.hpp"
00011
00012 using namespace std;
00013
00015 class TestParser {
00016 public:
00017     TestParser();
00018     ~TestParser();
00019     LocalModels* parse(string fileName);
00020
00021 protected:
00022     // @internal
00023 };
00024
00025 #endif
```

6.13 Types.cc File Reference

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

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

6.13.1 Detailed Description

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

6.14 Types.hpp File Reference

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

```
#include <map>
#include <set>
#include <stack>
#include <string>
#include <utility>
#include <vector>
```

Classes

- struct Var

  Represents a variable in the model, containing name, initial value and persistence.
- struct Condition

  Represents a condition for LocalTransition.
- struct Formula

  Contains a coalition of Agent from the formula.
- class Agent

  Contains all data for a single Agent, including id, name and all of the agents' variables.
- class LocalState

  Represents a single LocalState, containing id, name and internal variables.
- struct VarAssignment
- struct LocalTransition

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

  Represents a single local model, contains all agents and variables.
- struct GlobalModel

  Represents a global model, containing agents and a formula.
- struct GlobalState

  Represents a single global state.
- struct GlobalTransition

  Represents a single global transition.
- struct EpistemicClass

  Represents a single epistemic class.

Enumerations

- enum ConditionOperator { Equals , NotEquals }

  Conditional operator for the variable.
- enum GlobalStateVerificationStatus { UNVERIFIED , PENDING , VERIFIED_OK , **VERIFIED_ERR** }

  Verification status of a GlobalState.
- enum VarAssignmentType { FromVar , FromValue }

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

6.14.1 Detailed Description

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

6.14.2 Enumeration Type Documentation

6.14.2.1 ConditionOperator

```
enum ConditionOperator
```

Conditional operator for the variable.

Enumerator

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

6.14.2.2 GlobalStateVerificationStatus

`enum GlobalStateVerificationStatus`

Verification status of a GlobalState.

Enumerator

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

6.14.2.3 VarAssignmentType

`enum VarAssignmentType`

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

Enumerator

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

6.15 Types.hpp

Go to the documentation of this file.

```
00001
00007 #ifndef SELENE_TYPES
00008 #define SELENE_TYPES
00009
00010 #include <map>
00011 #include <set>
00012
00013 #include <stack>
00014 #include <string>
00015 #include <utility>
00016 #include <vector>
00017
00018 using namespace std;
00019
00020
00021 class Agent;
00022 class LocalState;
00023
```

```
00024 struct Condition;
00025 struct EpistemicClass;
00026 struct Formula;
00027 struct GlobalModel;
00028 struct GlobalState;
00029 struct GlobalTransition;
00030 struct LocalTransition;
00031 struct LocalModels;
00032 struct Var;
00033 struct VarAssignment;
00034
00036 enum ConditionOperator {
00037     Equals,
00038     NotEquals,
00039 };
00040
00042 enum GlobalStateVerificationStatus {
00043     UNVERIFIED,
00044     PENDING,
00045     VERIFIED_OK,
00046     VERIFIED_ERR,
00047 };
00048
00050 enum VarAssignmentType {
00051     FromVar,
00052     FromValue,
00053 };
00054
00056 struct Var {
00058     string name;
00059
00061     int initialValue;
00062
00064     bool persistent;
00065
00067     Agent *agent;
00068 };
00069
00071 struct Condition {
00073     Var* var;
00074
00076     ConditionOperator conditionOperator;
00077
00079     int comparedValue;
00080 };
00081
00083 struct Formula {
00085     set<Agent*> coalition;
00086 };
00087
00089 class Agent {
00090     public:
00092         int id;
00093
00095         string name;
00096
00098         set<Var*> vars;
00099
00103         Agent(int _id, string _name):id(_id), name(_name) {};
00104
00106         LocalState* initState;
00107
00109         vector<LocalState*> localStates; // localStates[i].id == i
00110
00112         vector<LocalTransition*> localTransitions; // localTransitions[i].id == i
00113
00114         // sprawdź, czy stan nie został już wygenerowany.
00115
00116         LocalState* includesState(LocalState *state);
00117 };
00118
00120 class LocalState {
00121     public:
00122         // Data
00123
00125         int id;
00126
00128         string name;
00129
00131         map<Var*, int> vars;
00132
00133         // alternatywna wersja - może wystarczy
00134
00136         map<string, int> environment;
00137
00138         // komparator
00139
```

```
00140         bool compare(LocalState *state);
00141
00142         // Bindings
00143
00145         Agent* agent;
00146
00148         set<LocalTransition*> localTransitions;
00149 };
00150
00151 // to jest zbędne
00152 struct VarAssignment {
00153     Var* dstVar;
00154     VarAssignmentType type; // zbędne
00155     Var* srcVar;
00156     int value;
00157 };
00158
00160 struct LocalTransition {
00161     // Data
00162
00164     int id;
00165
00167     string name;
00168
00170     string localName;
00171     // if empty => same as name
00172
00174     bool isShared;
00175
00177     int sharedCount;
00178
00180     set<Condition*> conditions;
00181
00183     set<VarAssignment*> varAsssignments;
00184
00185     // Bindings
00186
00188     Agent* agent;
00189
00191     LocalState* from;
00192
00194     LocalState* to;
00195
00197     set<LocalTransition*> sharedLocalTransitions;
00198 };
00199
00201 struct LocalModels {
00203     vector<Agent*> agents;
00204     // agents[i].id == i
00205
00207     map<string, Var*> vars;
00208     // vars[str].name == str
00209 };
00210
00212 struct GlobalModel {
00213     // Data
00214
00216     vector<Agent*> agents;
00217     // agents[i].id == i
00218
00220     Formula* formula;
00221
00222     // Bindings
00223
00225     GlobalState* initState;
00226
00228     vector<GlobalState*> globalStates;
00229     // globalStates[i].id == i
00230
00232     vector<GlobalTransition*> globalTransitions;
00233     // globalTransitions[i].id == i
00234
00236     map<Agent*, map<string, EpistemicClass*» epistemicClasses;
00237     // Agent* => (EpistemicClass->hash => EpistemicClass*)
00238 };
00239
00241 struct GlobalState {
00242     // Data
00243
00245     int id;
00246
00248     string hash;
00249
00251     map<Var*, int> vars;
00252
00254     map<Agent*, EpistemicClass*> epistemicClasses;
00255
```

```
00257     bool isExpanded;
00258
00260     GlobalStateVerificationStatus verificationStatus;
00261
00262     // Bindings
00263
00265     set<GlobalTransition*> globalTransitions;
00266
00268     set<LocalState*> localStates;
00269 };
00270
00272 struct GlobalTransition {
00273     // Data
00274
00276     int id;
00277
00279     bool isInvalidDecision;
00280
00281     // Bindings
00282
00284     GlobalState* from;
00285
00287     GlobalState* to;
00288
00290     set<LocalTransition*> localTransitions;
00291 };
00292
00294 struct EpistemicClass {
00296     string hash;
00297
00299     map<string, GlobalState*> globalStates;
00300     // GlobalState->hash => GlobalState*
00301
00303     GlobalTransition* fixedCoalitionTransition;
00304 };
00305
00306 #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.

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

6.19 Verification.cpp File Reference

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

```
#include "Verification.hpp"
```

Functions

- string verStatusToStr (GlobalStateVerificationStatus status)

  Converts global verification status into a string.
- void dbgVerifStatus (string prefix, GlobalState *gs, GlobalStateVerificationStatus st, string reason)

  Print a debug message of a verification status to the console.
- void dbgHistEnt (string prefix, HistoryEntry *h)

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

Variables

- Cfg **config**

6.19.1 Detailed Description

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

6.19.2 Function Documentation

6.19.2.1 dbgHistEnt()

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

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

Parameters

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

  Stores history and allows displaying it to the console.
- class Verification

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

Enumerations

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

  HistoryEntry entry type.
- enum Mode { NORMAL , REVERT , RESTORE }

  Current model traversal mode.

Functions

- string verStatusToStr (GlobalStateVerificationStatus status)

  Converts global verification status into a string.

6.20.1 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.

```
00001
00005 #ifndef SELENE_VERIFICATION
00006 #define SELENE_VERIFICATION
00007
00008 #include <stack>
00009 #include "Types.hpp"
00010 #include "GlobalModelGenerator.hpp"
00011 #include "SeleneFormula.hpp"
00012
00013 string verStatusToStr(GlobalStateVerificationStatus status);
00014
00016 enum HistoryEntryType {
00017     DECISION,
00018     STATE_STATUS,
00019     CONTEXT,
00020     MARK_DECISION_AS_INVALID,
00021 };
00022
00024 struct HistoryEntry {
00026     HistoryEntryType type;
00028     GlobalState* globalState;
00030     GlobalTransition* decision;
00032     bool globalTransitionControlled;
00034     GlobalStateVerificationStatus prevStatus;
00036     GlobalStateVerificationStatus newStatus;
00038     int depth;
00040     HistoryEntry* prev;
00042     HistoryEntry* next;
00045     string toString() {
00046         char buff[1024] = { 0 };
00047         if (this->type == HistoryEntryType::DECISION) {
00048             snprintf(buff, sizeof(buff), "decision in %s: to %s", this->globalState->hash.c_str(),
      this->decision->to->hash.c_str());
00049         }
00050         else if (this->type == HistoryEntryType::STATE_STATUS) {
00051             snprintf(buff, sizeof(buff), "stateVerStatus of %s: %s -> %s",
      this->globalState->hash.c_str(), verStatusToStr(this->prevStatus).c_str(),
      verStatusToStr(this->newStatus).c_str());
00052         }
00053         else if (this->type == HistoryEntryType::CONTEXT) {
00054             snprintf(buff, sizeof(buff), "context in %s at depth %i: to %s (%s)",
      this->globalState->hash.c_str(), this->depth, this->decision->to->hash.c_str(),
      this->globalTransitionControlled ? "controlled" : "uncontrolled");
00055         }
00056         else if (this->type == HistoryEntryType::MARK_DECISION_AS_INVALID) {
00057             snprintf(buff, sizeof(buff), "markInvalid in %s: to %s", this->globalState->hash.c_str(),
      this->decision->to->hash.c_str());
00058         }
00059         return string(buff);
00060     };
00061 };
00062
00064 class HistoryDbg {
00065 public:
00067     vector<pair<HistoryEntry*, char» entries;
00068     HistoryDbg();
00069     ~HistoryDbg();
00070     void addEntry(HistoryEntry* entry);
00071     void markEntry(HistoryEntry* entry, char chr);
00072     void print(string prefix);
00073     HistoryEntry* cloneEntry(HistoryEntry* entry);
00074 };
00075
00076 // On-the-fly traversal mode
00078 enum Mode {
00079     NORMAL,
```

```
00080     REVERT,
00081     RESTORE,
00082 };
00083
00085 class Verification {
00086 public:
00087     Verification(GlobalModelGenerator* generator);
00088     ~Verification();
00089     bool verify();
00090 protected:
00092     Mode mode;
00094     GlobalState* revertToGlobalState;
00096     stack<HistoryEntry*> historyToRestore;
00098     GlobalModelGenerator* generator;
00100     SeleneFormula* seleneFormula;
00102     HistoryEntry* historyStart;
00104     HistoryEntry* historyEnd;
00105     bool verifyLocalStates(set<LocalState*>* localStates);
00106     bool verifyGlobalState(GlobalState* globalState, int depth);
00107     bool isGlobalTransitionControlledByCoalition(GlobalTransition* globalTransition);
00108     bool isAgentInCoalition(Agent* agent);
00109     EpistemicClass* getEpistemicClassForGlobalState(GlobalState* globalState);
00110     bool areGlobalStatesInTheSameEpistemicClass(GlobalState* globalState1, GlobalState* globalState2);
00111     void addHistoryDecision(GlobalState* globalState, GlobalTransition* ecision);
00112     void addHistoryStateStatus(GlobalState* globalState, GlobalStateVerificationStatus prevStatus,
      GlobalStateVerificationStatus newStatus);
00113     void addHistoryContext(GlobalState* globalState, int depth, GlobalTransition* decision, bool
      globalTransitionControlled);
00114     void addHistoryMarkDecisionAsInvalid(GlobalState* globalState, GlobalTransition* decision);
00115     HistoryEntry* newHistoryMarkDecisionAsInvalid(GlobalState* globalState, GlobalTransition*
      decision);
00116     bool revertLastDecision(int depth);
00117     void undoLastHistoryEntry(bool freeMemory);
00118     void undoHistoryUntil(HistoryEntry* historyEntry, bool inclusive, int depth);
00119     void printCurrentHistory(int depth);
00120 };
00121
00122 #endif // SELENE_VERIFICATION
```

Index