Parsybone

1.0

Generated by Doxygen 1.7.6.1

Sat Sep 22 2012 23:38:19

Contents

Chapter 1

Welcome to the Parsybone code documentation.

This is a documentation of code belonging solely to the Parsybone tool, version 1.0. This text is not supposed to be a user manual in any way, but as an reference for further development of this tool. For a description of usage of the tool, please refer to the Manual that is shipped together with the tool.

This file is part of ParSyBoNe (Parameter Synthetizer for Boolean Networks) verification tool.

This software has been created as a part of a research conducted in the Systems - Biology Laboratory of Masaryk University Brno. See http://sybila.fi.muni.cz/.

Chapter 2

Namespace Index

2.1 Namespace List

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

Chapter 3

Class Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

Namespace Documentation

5.1 Translator Namespace Reference

Methods used for translation of string data to variables during model parsing.

5.2 XMLHelper Namespace Reference

This namespace encapsulates simple parsing functions with tests for bigger robustness.

Chapter 6

Class Documentation

6.1 ArgumentParser Class Reference

A class responsible for reading the arguments on the input.

`#include <argument_parser.hpp>`

Public Member Functions

- void parseArguments (const std::vector< std::string > &arguments, std::ifstream &input_stream)

6.1.1 Detailed Description

A sets user options according to the string provided as arguments at the start of the program. All values that are not used for direct setup are stored within a UserOptions class.

6.1.2 Member Function Documentation

6.1.2.1 void **ArgumentParser::parseArguments** (const std::vector< std::string > & ***arguments,*** std::ifstream & ***input_stream***) `[inline]`

Take all the arguments on the input and store information from them.

Parameters

argc	passed from main function
argv	passed from main function
intput_-stream	pointer to a file that will be used as an input stream

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

- parsing/argument_parser.hpp

6.2 AutomatonBuilder Class Reference

Transform graph of the automaton into a set of labeled transitions in an AutomatonStructure object.

```
#include <automaton_builder.hpp>
```

Public Member Functions

- AutomatonBuilder (const Model &_model, AutomatonStructure &_automaton)
- void buildAutomaton ()

6.2.1 Detailed Description

This builder creates a basic automaton controlling property - this automaton is based on the AutomatonInterface. Automaton is provided with string labels on the edges that are parsed and resolved for the graph.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 AutomatonBuilder::AutomatonBuilder (const **Model** & *_model,* **AutomatonStructure** & *_automaton*) [inline]

Constructor computes boundaries of the state space and passes references.

6.2.3 Member Function Documentation

6.2.3.1 void **AutomatonBuilder::buildAutomaton** ( ) [inline]

Create the transitions from the model and fill the automaton with them.

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

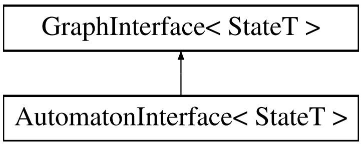
- construction/automaton_builder.hpp

6.3 AutomatonInterface< StateT > Class Template Reference

Interface for all the classes that represent a Buchi automaton. Buchi automaton is based on a GraphInterface.

```
#include <automaton_interface.hpp>
```

Inheritance diagram for AutomatonInterface< StateT >:



Public Member Functions

- virtual bool isFinal (const StateID ID) const
- virtual bool isInitial (const StateID ID) const
- virtual const std::vector < StateID > & getFinalStates () const
- virtual const std::vector < StateID > & getInitialStates () const

Protected Attributes

- std::vector< StateID > initial_states

  Vector with indexes of initial states (in this case only the first state).
- std::vector< StateID > final_states

  Vector with indexes of final states of the BA.

template< typename StateT > class AutomatonInterface< StateT >

6.3.1 Member Function Documentation

6.3.1.1 template< typename StateT > virtual const std::vector< StateID >& AutomatonInterface< StateT >::getFinalStates ( ) const [inline, virtual]

Get IDs of all states that are marked as final.

Returns

vector of final states' IDs

6.3.1.2 template< typename StateT > virtual const std::vector< StateID >& AutomatonInterface< StateT >::getInitialStates ( ) const [inline, virtual]

Get IDs of all states that are marked as initial.

Returns

vector of initial states' IDs

6.3.1.3 template<typename StateT> virtual bool AutomatonInterface< StateT >::isFinal (const StateID *ID*) const `[inline, virtual]`

For a given state find out whether it is marked as final.

Parameters

ID	state to test

Returns

true if the state is final

6.3.1.4 template<typename StateT> virtual bool AutomatonInterface< StateT >::isInitial (const StateID *ID*) const `[inline, virtual]`

For given state find out if it is marked as initial.

Parameters

ID	state to test

Returns

true if the state is initial

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

- construction/automaton_interface.hpp

6.4 AutomatonParser Class Reference

This object is responsible for parsing and translation of data related to the tested property.

`#include <automaton_parser.hpp>`

Public Member Functions

- AutomatonParser (Model &_model)

  Simple constructor, passes references.
- void parse (const rapidxml::xml_node<> ∗const model_node)

6.4.1 Member Function Documentation

6.4.1.1 void AutomatonParser::parse (const rapidxml::xml_node<> *const *model_node*) [inline]

Main parsing function. It expects a pointer to inside of a MODEL node.

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

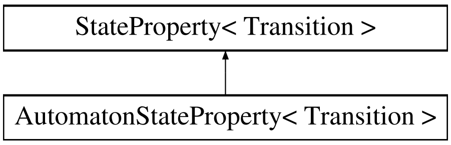
- parsing/automaton_parser.hpp

6.5 AutomatonStateProperty< Transition > Struct Template - Reference

A state structure enhanced with information whether the state is final and/or initial.

`#include <automaton_interface.hpp>`

Inheritance diagram for AutomatonStateProperty< Transition >:


StateProperty< Transition >
AutomatonStateProperty< Transition >

Public Member Functions

- AutomatonStateProperty (const bool _initial, const bool _final, const StateID ID, const std::string &&label)

Public Attributes

- bool initial

  True if the state is initial.
- bool final

  True if this state is final.

template<typename Transition> struct AutomatonStateProperty< Transition >

6.5.1 Constructor & Destructor Documentation

6.5.1.1 template<typename Transition> AutomatonStateProperty< Transition >::AutomatonStateProperty (const bool *_initial*, const bool *_final*, const StateID *ID*, const std::string && *label*) [inline]

Adds information if the state is final or initial, passes the rest.

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

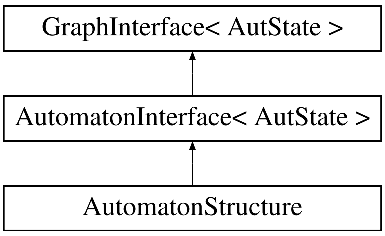
- construction/automaton_interface.hpp

6.6 AutomatonStructure Class Reference

A Buchi automaton designed to control some ω-regular property.

`#include <automaton_structure.hpp>`

Inheritance diagram for AutomatonStructure:



Public Member Functions

- AutomatonStructure ()

  Default empty constructor.
- bool isTransitionFeasible (const StateID ID, const std::size_t transition_num, const Levels &levels) const

Friends

- class **AutomatonBuilder**

6.6.1 Detailed Description

AutomatonStructure stores Buchi automaton with edges labelled by values the KS can be in for the transition to be allowed. AutomatonStructure data can be set only from the AutomatonStructureBuilder object.

6.6.2 Member Function Documentation

6.6.2.1 bool AutomatonStructure::isTransitionFeasible (const StateID *ID*, const std::size_t *transition_num*, const Levels & *levels*) const [inline]

Checks if a transition of the BA is possible in the current state of a KS.

Parameters

ID	source state of the transition
transition_-num	ordinal number of the transition
levels	current levels of species i.e. the state of the KS

Returns

true if the transition is feasible

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

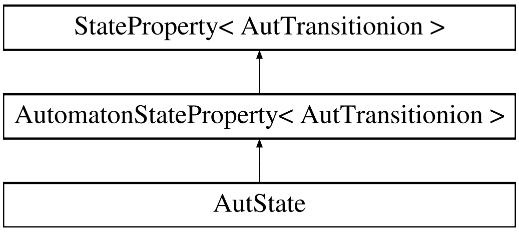
- construction/automaton_structure.hpp

6.7 AutState Struct Reference

Storing a single state of the Buchi automaton. This state is extended with a value saying wheter the states is final.

`#include <automaton_structure.hpp>`

Inheritance diagram for AutState:


StateProperty< AutTransitionion >
AutomatonStateProperty< AutTransitionion >
AutState

Public Member Functions

- AutState (const StateID ID, const bool final, std::string &&label)

  Fills data and checks if the state has value -> is initial.

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

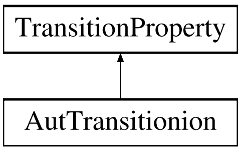
- construction/automaton_structure.hpp

6.8 AutTransitionion Struct Reference

Single labelled transition from one state to another.

`#include <automaton_structure.hpp>`

Inheritance diagram for AutTransitionion:



Public Member Functions

- AutTransitionion (const StateID target_ID, AllowedValues &&_allowed_values)

  Simple filler, assigns values to all the variables.

Public Attributes

- AllowedValues allowed_values

  Allowed values of species for this transition.

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

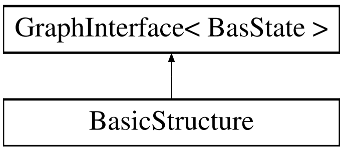
- construction/automaton_structure.hpp

6.9 BasicStructure Class Reference

A simple structure describing the complete state space.

`#include <basic_structure.hpp>`

Inheritance diagram for BasicStructure:



Public Member Functions

- BasicStructure ()

Default empty constructor, needed to create an empty object that will be filled.

- const Levels & getStateLevels (const StateID ID) const
- std::size_t getSpecieID (const StateID ID, const std::size_t neighbour_index) const
- Direction getDirection (const StateID ID, const std::size_t neighbour_index) const

Friends

- class **BasicStructureBuilder**

6.9.1 Detailed Description

BasicStructure stores states of the Kripke structure created from the model - each state knows its levels and indexes of all the neighbours. Order of neighbours of state is (specie 1 down, specie 1 stay, specie 1 up, specie 2 down, ...) BasicStructure data can be set only form the BasicStructureBuilder object.

6.9.2 Member Function Documentation

6.9.2.1 Direction **BasicStructure::getDirection** (const StateID *ID*, const std::size_t *neighbour_index*) const [inline]

Parameters

ID	ID of the state to get the neighbour from
neighbour_index	index in the vector of neighbours

Returns

Direction in which the specie changes

6.9.2.2 std::size_t **BasicStructure::getSpecieID** (const StateID *ID*, const std::size_t *neighbour_index*) const [inline]

Parameters

ID	ID of the state to get the neighbour from
neighbour_index	index in the vector of neighbours

Returns

ID of the specie that vary between the two states

6.9.2.3 const Levels& BasicStructure::getStateLevels (const StateID *ID*) const [inline]

Parameters

ID	ID of the state to get

Returns

levels of the state

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

- construction/basic_structure.hpp

6.10 BasicStructureBuilder Class Reference

Creates a full state space as a simple graph as a BasicStructure object.

```
#include <basic_structure_builder.hpp>
```

Public Member Functions

- BasicStructureBuilder (const Model &_model, BasicStructure &_structure)
- void buildStructure ()

6.10.1 Detailed Description

BasicStructureBuilder creates the BasicStructure (Simple Kripke Structure) from the model data. In each iteration of the creation, a new state is generated as a cartesian product of values of the species. All the combinations are used. Each state is provided with indexes of their neighbours. For each dimension (specie) there are three neighbours, if possible, base on the change of the specie's value - up, stay or down.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 BasicStructureBuilder::BasicStructureBuilder (const Model & *_model,* BasicStructure & *_structure*) [inline]

Constructor initializes basic information from the model

6.10.3 Member Function Documentation

6.10.3.1 void BasicStructureBuilder::buildStructure ( ) [inline]

Create the states from the model and fill the structure with them.

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

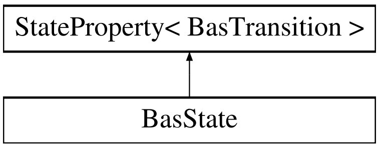
- construction/basic_structure_builder.hpp

6.11 BasState Struct Reference

Storing a single state - its activation levels of each of the species and IDs of states that are neighbours (differ only in single step of single value).

```
#include <basic_structure.hpp>
```

Inheritance diagram for BasState:



Public Member Functions

- BasState (const StateID ID, const Levels _species_level, const std::string &&label)

    Simple filler, assigns values to all the variables.

Public Attributes

- Levels species_level

    Species_level[i] = activation level of specie i.

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

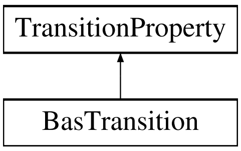
- construction/basic_structure.hpp

6.12 BasTransition Struct Reference

Stores an unlabelled transition to next state.

```
#include <basic_structure.hpp>
```

Inheritance diagram for BasTransition:



Public Member Functions

- BasTransition (const StateID target_ID, const std::size_t _changed_specie, const Direction _change_direction)

  Simple filler, assigns values to all the variables.

Public Attributes

- std::size_t changed_specie

  ID of specie that differs between this and neighbour.
- Direction change_direction

  Way the specie's value is changed.

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

- construction/basic_structure.hpp

6.13 ColoringAnalyzer Class Reference

A storage of individual final states together with their coloring and to further provide necessary data for the output as requested.

```
#include <coloring_analyzer.hpp>
```

Public Member Functions

- const std::vector< std::string > getOutput () const
- const std::vector< std::string > getStrings (const StateID ID) const
- const std::vector< std::string > getStrings () const
- const std::vector< ColorNum > getNumbers (const StateID ID) const
- const std::vector< ColorNum > getNumbers () const
- Paramset getMask (const StateID ID) const
- Paramset getMask () const

Friends

- class **SynthesisManager**

6.13.1 Member Function Documentation

6.13.1.1 Paramset ColoringAnalyzer::getMask (const StateID *ID*) const [inline]

Parameters

ID	index of the state to get the mask from

Returns

coloring of the given state or 0 if the state is not present

6.13.1.2 Paramset ColoringAnalyzer::getMask ( ) const [inline]

Compute merge of all final colors, creating a coloring with all feasible colors in this round.

Returns

all feasible colors in this round

6.13.1.3 const std::vector<ColorNum> ColoringAnalyzer::getNumbers (const StateID *ID*) const [inline]

Parameters

ID	index of the state to get the mask from

Returns

ordinal number of the parametrizations that are acceptable

6.13.1.4 const std::vector<ColorNum> ColoringAnalyzer::getNumbers ( ) const [inline]

Returns

ordinal numbers of the parametrizations that are acceptable in this round

6.13.1.5 const std::vector<std::string> ColoringAnalyzer::getOutput ( ) const [inline]

Computes a vector of strings of acceptable colors from this round with their output, as requested by user.

Returns

vector of strings with numbers of parametrizations and their explicit form

6.13.1.6 const std::vector<std::string> ColoringAnalyzer::getStrings (const StateID *ID*) const [inline]

Obtain colors given parameters in the form [fun1, fun2, ...] for required state.

Parameters

ID	index of the state to get the mask from

Returns

vector of numbers and strings of colors

6.13.1.7 const std::vector<std::string> ColoringAnalyzer::getStrings ( ) const [inline]

Obtain colors given parameters in the form [fun1, fun2, ...] for all parameters in this round.

Returns

vector of numbers and strings of colors

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

- synthesis/coloring_analyzer.hpp

6.14 ColoringParser Class Reference

Parser for the bitmask of controlled parametrizations.

```
#include <coloring_parser.hpp>
```

Public Member Functions

- ColoringParser ()

  Ddefault constructor.
- void openFile (const std::string filename)
- void createOutput (const std::string filename)
- void parseMask ()
- void outputComputed (const Paramset parameters)
- const std::vector< Paramset > & getColors () const
- std::size_t getColorsCount ()

6.14.1 Detailed Description

Coloring parser reads a bitmask of colors from the file.

Attention

Mask file always needs to have number of bytes dividable by Parameters size. If the last set is smaller, it must be shifted to the right!

6.14.2 Member Function Documentation

6.14.2.1 void **ColoringParser::createOutput** (const std::string *filename*) [inline]

Create a file to output bitmasks to.

Parameters

filename	path to the file to read from

6.14.2.2 const std::vector<Paramset>& **ColoringParser::getColors** ( ) const [inline]

Returns

masks for all colors that can be used

6.14.2.3 std::size_t **ColoringParser::getColorsCount** ( ) [inline]

Returns

number of Parameters e.g. number of rounds of computation

6.14.2.4 void **ColoringParser::openFile** (const std::string *filename*) [inline]

Only opens the file with the data stream.

Parameters

filename	path to the file to read from

6.14.2.5 void **ColoringParser::outputComputed** (const Paramset *parameters*) [inline]

Send computed data for this round on the ouput.

Parameters

parameters	bitmask of computed feasible colors

6.14.2.6 void ColoringParser::parseMask ( ) [inline]

Main parsing function that creates parameters vector.

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

- parsing/coloring_parser.hpp

6.15 ColorStorage Class Reference

An auxiliary class to the ProductStructure and stores colors and possibly predecessors for individual states of the product during the computation.

```
#include <color_storage.hpp>
```

Classes

- struct **State**

Public Member Functions

- ColorStorage (const ConstructionHolder &holder)
- ColorStorage ()

  Empty constructor for an empty storage.
- void addFrom (const ColorStorage &other)
- void reset ()
- void setResults (const std::vector< std::size_t > &new_cost, const Paramset resulting)
- void setResults (const Paramset resulting)
- bool update (const StateID ID, const Paramset parameters)
- bool soft_update (const StateID ID, const Paramset parameters)
- bool update (const StateID source_ID, const StateID target_ID, const Paramset parameters)
- void remove (const StateID ID, const Paramset remove)
- void remove (const StateID source_ID, const Paramset remove, const bool successors)
- void remove (const StateID source_ID, const StateID target_ID, const Paramset remove, const bool successors)
- std::size_t getMaxDepth () const
- const Paramset & getColor (const StateID ID) const

- const std::vector< Coloring > getColor (const std::vector< StateID > &states) const
- const Neighbours getNeighbours (const StateID ID, const bool successors, const Paramset color_mask=∼0) const
- const std::vector< Paramset > getMarking (const StateID ID, const bool successors, const Paramset color_mask=∼0) const
- std::size_t getCost (std::size_t position) const
- const std::vector< std::size_t > & getCost () const
- const Paramset & getAcceptable () const

6.15.1 Constructor & Destructor Documentation

6.15.1.1 ColorStorage::ColorStorage (const ConstructionHolder & *holder*) `[inline]`

Constructor allocates necessary memory for further usage (this memory is not supposed to be freed until endo of the computation). Every state has predecessors and succesors allocated for EVERY other state, this consumes memory but benefits the complexity of operations.

Parameters

states_count	number of states the structure the data will be saved for has

6.15.2 Member Function Documentation

6.15.2.1 void ColorStorage::addFrom (const ColorStorage & *other*) `[inline]`

Function adds values from specified source without explicitly copying them, only through bitwise or (storages must be equal).

6.15.2.2 const Paramset& ColorStorage::getAcceptable ( ) const `[inline]`

Returns

mask of parametrizations that are computed acceptable in this round

6.15.2.3 const Paramset& ColorStorage::getColor (const StateID *ID*) const `[inline]`

Parameters

ID	index of the state to ask for parameters

Returns

parameters assigned to the state

6.15.2.4 const std::vector< Coloring > ColorStorage::getColor (const std::vector< StateID > & *states*) const [inline]

Parameters

states	indexes of states to ask for parameters

Returns

queue with all colorings of states

6.15.2.5 std::size_t ColorStorage::getCost (std::size_t *position*) const [inline]

Parameters

number	of the parametrization relative in this round

Returns

Cost value of a particular parametrization

6.15.2.6 const std::vector< std::size_t >& ColorStorage::getCost ( ) const [inline]

Returns

Cost value of all the parametrizations from this round

6.15.2.7 const std::vector< Paramset > ColorStorage::getMarking (const StateID *ID*, const bool *successors*, const Paramset *color_mask* = ∼0) const [inline]

Get all the labels on trasintions from given neighbours.

Parameters

ID	index of the state to ask for predecessors
successors	true if successors are required, false if predecessors
color_mask	if specified, restricts neighbour to only those that contain a subset of the parametrizations

Returns

labelling on the neighbour labels

6.15.2.8 std::size_t ColorStorage::getMaxDepth ( ) const [inline]

Returns

max finite cost among parametrizations used this round

6.15.2.9 const Neighbours ColorStorage::getNeighbours (const StateID *ID*, const bool *successors*, const Paramset *color_mask* = ∼0) const [inline]

Get all the neigbours for this color from this state.

Parameters

ID	index of the state to ask for predecessors
successors	true if successors are required, false if predecessors
color_mask	if specified, restricts neighbour to only those that contain a subset of the parametrizations

Returns

neigbours for given state

6.15.2.10 void ColorStorage::remove (const StateID *ID*, const Paramset *remove*) [inline]

Removes given paramset from the coloring of the given state.

6.15.2.11 void ColorStorage::remove (const StateID *source_ID*, const Paramset *remove*, const bool *successors*) [inline]

Removes given paramset from the label of transitions to successors / from predecessors for a given state.

Parameters

successors	if true, use successors, predecessors otherwise

6.15.2.12 void ColorStorage::remove (const StateID *source_ID*, const StateID *target_ID*, const Paramset *remove*, const bool *successors*) [inline]

Removes given paramset from the label of transitions to a single successor / from a single predecessor for a given state.

Parameters

target_ID	ID of the state target state to remove parameset from labelling
successors	if true, use successors, predecessors otherwise

6.15.2.13 void ColorStorage::reset ( ) [inline]

Sets all values for all the states to zero. Allocated memory remains.

6.15.2.14 void ColorStorage::setResults (const std::vector< std::size_t > & *new_cost*, const Paramset *resulting*) [inline]

Fills after time series check finished.

Parameters

new_cost	a vector of lenght |parameter_set| containing cost values. If the value does not exist (state is not reachable), use ∼0

6.15.2.15 void ColorStorage::setResults (const Paramset *resulting*) [inline]

Fills after a general LTL check finished.

Parameters

new_cost	a vector of lenght |parameter_set| containing cost values. If the value does not exist (state is not reachable), use ∼0

6.15.2.16 bool ColorStorage::soft_update (const StateID *ID*, const Paramset *parameters*) [inline]

Return true if the state would be updated, false otherwise.

Parameters

ID	index of the state to fill
parameters	to add - if empty, add all, otherwise use bitwise or

Returns

true if there would be an update

6.15.2.17 bool **ColorStorage::update** (const StateID *ID*, const Paramset *parameters*) [inline]

Add passed colors to the state.

Parameters

ID	index of the state to fill
parameters	to add - if empty, add all, otherwise use bitwise or

Returns

true if there was an actuall update

6.15.2.18 bool **ColorStorage::update** (const StateID *source_ID*, const StateID *target_ID*, const Paramset *parameters*) [inline]

Add passed colors to the state.

Parameters

source_ID	index of the state that passed this update
target_ID	index of the state to fill
parameters	to add - if empty, add all, otherwise use bitwise or

Returns

true if there was an actuall update

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

- synthesis/color_storage.hpp

6.16 ConstructionHolder Class Reference

Stores pointers to all data objects created for the purpose of the synthesis.

```
#include <construction_holder.hpp>
```

Public Member Functions

- void **fillModel** (Model ∗_model)

- ConstructionHolder ()

  Default (empty) constructor.
- const AutomatonStructure & **getAutomatonStructure** () const
- const BasicStructure & **getBasicStructure** () const
- const ParametrizationsHolder & **getParametrizations** () const
- const Model & **getModel** () const
- const LabelingHolder & **getLabeling** () const
- const ParametrizedStructure & **getParametrizedStructure** () const
- const ProductStructure & **getProduct** () const

Friends

- class **ConstructionManager**

6.16.1 Detailed Description

Class stores and provides all the objects that are built during construction phase. There are two methods employed:

1. fill∗ this method obtains a reference for a dynamic object and assigns it to its unique_ptr,

get∗ this method returns a constant reference to a requested object.

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

- construction/construction_holder.hpp

6.17 ConstructionManager Class Reference

STEP 2 - Builds all the structures and stores them within a ConstructionHolder.

```
#include <construction_manager.hpp>
```

Public Member Functions

- ConstructionManager (ConstructionHolder &_holder)
- void construct ()

6.17.1 Detailed Description

ConstructionManager overviews the whole process of construction of structures from information contained within a model file. All the objects constructed are stored within a provided CostructionHolder and further acessible only via constant getters.

6.17.2 Constructor & Destructor Documentation

6.17.2.1 ConstructionManager::ConstructionManager (ConstructionHolder & *_holder*) [inline]

Constructor, passes the reference.

Parameters

_holder	object that will hold all the constructed objects

6.17.3 Member Function Documentation

6.17.3.1 void ConstructionManager::construct ( ) [inline]

Function that constructs all the data in a cascade of temporal builders.

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

- construction/construction_manager.hpp

6.18 FormulaeParser Class Reference

Class able to resolve any logical function in propositional logic.

`#include <formulae_parser.hpp>`

Static Public Member Functions

- static bool resolve (const std::map< std::string, bool > &valuation, std::string formula)

6.18.1 Detailed Description

This is a static helper class able of resolving any preposition logic formula. Formula construction:

1. tt (true) and ff (false) are formulas representing true and false respectively,
   (a) any variable is a formula,
2. for φ formula is $!\varphi$ formula,
3. for ψ,φ formulas are $(\psi|\varphi)$, $(\psi\&\varphi)$ formulas representing logical disjunction and conjunction respectively,
4. nothing else is a formula.

6.18.2 Member Function Documentation

6.18.2.1 static bool **FormulaeParser::resolve** (const std::map< std::string, bool > & *valuation*, std::string *formula*) [inline, static]

Function that returns valuation of the formula based on valuation of its variables.

Parameters

in	*valuation*	map of variable valuations in the form (name, value)
in	*formula*	formula to resolve

Returns

true iff valuation of the formula is true

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

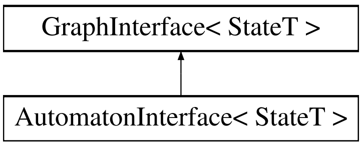
- parsing/formulae_parser.hpp

6.19 GraphInterface< StateT > Class Template Reference

Interface for all the classes that represent a directed graph. Transitions are expected to be stored within their source state structure.

`#include <graph_interface.hpp>`

Inheritance diagram for GraphInterface< StateT >:



Public Member Functions

- std::size_t getStateCount () const
- std::size_t getTransitionCount (const StateID ID) const
- StateID getTargetID (const StateID ID, const std::size_t transition_number) const
- const std::string & getString (const StateID ID) const

Protected Attributes

- std::vector< StateT > states

  Vector holding states of the graph.

template<typename StateT> class GraphInterface< StateT >

6.19.1 Member Function Documentation

6.19.1.1 template<typename StateT> std::size_t GraphInterface< StateT >::getStateCount ( ) const [inline]

Obtains number of states of the graph.

Returns

integer with size of the graph

6.19.1.2 template<typename StateT> const std::string& GraphInterface< StateT >::getString (const StateID *ID*) const [inline]

Returns given state as a string.

Parameters

ID	ID of the state to turn into the string

Returns

given state as a string

6.19.1.3 template<typename StateT> StateID GraphInterface< StateT >::getTargetID (const StateID *ID,* const std::size_t *transition_number*) const [inline]

Obtains ID of the target of given transition for given state.

Parameters

ID	ID of the state to get the neighbour from
trans_-number	index in the vector of transitions

Returns

ID of the requested target

6.19.1.4 template<typename StateT> std::size_t GraphInterface< StateT >::getTransitionCount (const StateID *ID*) const [inline]

Obtains number of outcoming transitions for given state.

Parameters

ID	ID of the state to get the number from

Returns

integer with number of outcoming transitions

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

- construction/graph_interface.hpp

6.20 LabelingBuilder Class Reference

Creates a labeled graph representation of gene regulatory network and stores it within a LabelingHolder object.

```
#include <labeling_builder.hpp>
```

Public Member Functions

- LabelingBuilder (const Model &_model, const ParametrizationsHolder &_parametrizations, LabelingHolder &_labeling_holder)
- void buildLabeling ()

6.20.1 Constructor & Destructor Documentation

6.20.1.1 LabelingBuilder::LabelingBuilder (const **Model** & *_model,* const **ParametrizationsHolder** & *_parametrizations,* **LabelingHolder** & *_labeling_holder*) [inline]

Constructor just attaches the references to data holders

6.20.2 Member Function Documentation

6.20.2.1 void **LabelingBuilder::buildLabeling** ( ) [inline]

For each specie recreate all its regulatory functions (all possible labels)

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

- construction/labeling_builder.hpp

6.21 LabelingHolder Class Reference

Storage for the regulatory graph with kinetic parameters encoded in form of regulatory functions.

`#include <labeling_holder.hpp>`

Classes

- struct **RegulatoryFunction**

  Storing a regulatory function in explicit form.
- struct **Specie**

  Storing a sigle specie with its regulations.

Public Member Functions

- LabelingHolder ()

  Default empty constructor.
- std::size_t getParametersCount () const
- std::size_t getSpeciesCount () const
- const std::string & getSpecieName (const std::size_t ID) const
- const std::vector< std::size_t > & getSpecieValues (const std::size_t ID) const
- const std::vector< std::size_t > & getSourceSpecies (const std::size_t ID) const
- std::size_t getRegulationsCount (const std::size_t ID) const
- std::size_t getStepSize (const std::size_t ID, const std::size_t regulation) const
- const std::vector< std::size_t > & getPossibleValues (const std::size_t ID, const std::size_t regulation) const
- const std::vector< std::vector < std::size_t > > & getSourceValues (const std::size_t ID, const std::size_t regulation) const

Friends

- class **LabelingBuilder**

6.21.1 Detailed Description

LabelingHolder contains basic representation of the Gene Regulatory network in the form of the labeled graph. Each specie is stored together with its regulations. Each regulation has its step_size value (shared by multiple regulations). This value represents division of parametrization space and is used for encoding and decoding it into paramset. LabelingHolder data can be set only form the LabelingBuilder object.

6.21.2 Member Function Documentation

6.21.2.1 std::size_t LabelingHolder::getParametersCount ( ) const [inline]

Returns

size of the parameter space

6.21.2.2 const std::vector<std::size_t>& LabelingHolder::getPossibleValues (const std::size_t *ID,* const std::size_t *regulation*) const [inline]

Returns

values this function can possibly regulate to

6.21.2.3 std::size_t LabelingHolder::getRegulationsCount (const std::size_t *ID*) const [inline]

Returns

number of regulations for this specie (two to power of number of source species)

6.21.2.4 const std::vector<std::size_t>& LabelingHolder::getSourceSpecies (const std::size_t *ID*) const [inline]

Returns

IDs of all the species that regulate this specie

6.21.2.5 const std::vector<std::vector<std::size_t> >& LabelingHolder::getSourceValues (const std::size_t *ID,* const std::size_t *regulation*) const [inline]

Returns

for each source specie all the values that if it is within them, it allows this function

6.21.2.6 const std::string& LabelingHolder::getSpecieName (const std::size_t *ID*) const [inline]

Returns

name of the specie with given ID

6.21.2.7 std::size_t LabelingHolder::getSpeciesCount ( ) const [inline]

Returns

number of the species

6.21.2.8 const std::vector<std::size_t>& LabelingHolder::getSpecieValues (const std::size_t *ID*) const [inline]

Returns

all the values the specie can occur in

6.21.2.9 std::size_t LabelingHolder::getStepSize (const std::size_t *ID*, const std::size_t *regulation*) const [inline]

Returns

step_size (how many neigbour parameters share the same value for this regulation)

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

- construction/labeling_holder.hpp

6.22 Model Class Reference

Storage for data parsed from the model.

`#include <model.hpp>`

Classes

- struct **AdditionalInformation**

  Structure that stores additional information about the model.
- struct **BuchiAutomatonState**

  Structure that holds data about a single state.
- struct **ModelSpecie**

  Structure that holds data about a single specie. Most of the data is equal to that in the model file.
- struct Regulation

  Structure that stores regulation of a specie by another one.

Public Types

- typedef std::pair< std::vector < bool >, int > Parameter

    Kinetic parameter of the specie (bitmask of active incoming regulations, target value)
- typedef std::pair< StateID, std::string > Egde

    Edge in Buchi Automaton (Target ID, edge label)

Public Member Functions

- Model ()

    Default empty constructor.
- std::size_t getSpeciesCount () const
- std::size_t getStatesCount () const
- SpecieID findID (const std::string &name) const
- SpecieID findNumber (const std::string &name) const
- const std::string & getName (const std::size_t ID) const
- std::size_t getMin (const std::size_t ID) const
- std::size_t getMax (const std::size_t ID) const
- std::size_t getBasal (const std::size_t ID) const
- const std::vector< Regulation > & getRegulations (const std::size_t ID) const
- const std::vector< Parameter > & getParameters (const std::size_t ID) const
- bool isFinal (const std::size_t ID) const
- const std::vector< Egde > & getEdges (const std::size_t ID) const

Friends

- class **AutomatonParser**
- class **ModelParser**
- class **NetworkParser**
- class **TimeSeriesParser**

6.22.1 Detailed Description

Model stores model data in the raw form, almost the same as in the model file itself. Model data can be set only form the ModelParser object. Rest of the code can access the data only via constant getters - once the data are parse, model remains constant.

6.22.2 Member Function Documentation

6.22.2.1 SpecieID Model::findID (const std::string & *name*) const [inline]

Finds numerical ID of the specie based on its name or ID string.

Returns

ID of the specie with the specified name if there is such, otherwise ∼0

6.22.2.2 SpecieID Model::findNumber (const std::string & *name*) const [inline]

Finds ordinal number of the BA state based on its name or number string.

Returns

number of the state with the specified name if there is such, otherwise ~0

6.22.2.3 std::size_t Model::getBasal (const std::size_t *ID*) const [inline]

Returns

basal value of the specie

6.22.2.4 const std::vector<Egde>& Model::getEdges (const std::size_t *ID*) const [inline]

Returns

edges of the state

6.22.2.5 std::size_t Model::getMax (const std::size_t *ID*) const [inline]

Returns

maximal value of the specie

6.22.2.6 std::size_t Model::getMin (const std::size_t *ID*) const [inline]

Returns

minimal value of the specie (always 0)

6.22.2.7 const std::string& Model::getName (const std::size_t *ID*) const [inline]

Returns

name of the specie

6.22.2.8 const std::vector<Parameter>& Model::getParameters (const std::size_t *ID*) const [inline]

Returns

kinetic parameters of the regulations of the specie

6.22.2.9 const std::vector< Regulation >& Model::getRegulations (const std::size_t *ID*) const [inline]

Returns

regulations of the specie

6.22.2.10 std::size_t Model::getSpeciesCount ( ) const [inline]

Returns

number of the species

6.22.2.11 std::size_t Model::getStatesCount ( ) const [inline]

Returns

number of the states

6.22.2.12 bool Model::isFinal (const std::size_t *ID*) const [inline]

Returns

true if the state is final

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

- parsing/model.hpp

6.23 ModelChecker Class Reference

Main class of the computation - responsible for the CMC procedure.

```
#include <model_checker.hpp>
```

Public Member Functions

- ModelChecker (const ConstructionHolder &holder, ColorStorage &_storage)
- void startColoring (const StateID ID, const Paramset parameters, const Range &_range)
- void startColoring (const Paramset parameters, const std::set< StateID > &_updates, const Range &_range)

6.23.1 Detailed Description

ModelChecker class solves the parameter synthesis problem by iterative transfer of feasible parametrizations from initial states to final ones. Functions in model checker use many supporting variables and therefore are quite long, it would not make sense to split them, though.

6.23.2 Constructor & Destructor Documentation

6.23.2.1 ModelChecker::ModelChecker (const ConstructionHolder & *holder,* ColorStorage & *_storage*) [inline]

Constructor, passes the data and sets up auxiliary storage.

6.23.3 Member Function Documentation

6.23.3.1 void ModelChecker::startColoring (const StateID *ID,* const Paramset *parameters,* const Range & *_range*) [inline]

Start a new coloring round for cycle detection from a single state.

Parameters

ID	ID of the state to start cycle detection from
parameters	starting parameters for the cycle detection
_range	range of parameters for this coloring round

6.23.3.2 void ModelChecker::startColoring (const Paramset *parameters,* const std::set< StateID > & *_updates,* const Range & *_range*) [inline]

Start a new coloring round for cycle detection from a single state.

Parameters

parameters	starting parameters to color the structure with
_updates	states that are will be scheduled for an update in this round
_range	range of parameters for this coloring round

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

- synthesis/model_checker.hpp

6.24 ModelParser Class Reference

Starting point of the model parsing.

`#include <model_parser.hpp>`

Public Member Functions

- ModelParser (Model &_model, std::ifstream *_input_stream)

  Simple constructor, passes references.
- void parseInput ()

6.24.1 Detailed Description

ModelParser is an entry point for parsing of a model file. Most of the parsing is done by dependent classes, ModelParser only sets the parsing up for further usage. For the reference on how to create a model see the manual/README.

6.24.2 Member Function Documentation

6.24.2.1 void **ModelParser::parseInput ( )** `[inline]`

Functions that causes the parser to read the input from the stream, parse it and store model information in the model object.

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

- parsing/model_parser.hpp

6.25 NetworkParser Class Reference

Class for parsing of the regulatory network.

`#include <network_parser.hpp>`

Public Member Functions

- NetworkParser (Model &_model)

  Simple constructor, passes references.
- void parse (const rapidxml::xml_node<> *const model_node)

6.25.1 Detailed Description

This object is responsible for parsing and translation of data related to the GRN. Most of the possible semantics mistakes are under control and cause exceptions.

6.25.2 Member Function Documentation

6.25.2.1 void NetworkParser::parse (const rapidxml::xml_node<> *const *model_node*) [inline]

Main parsing function. It expects a pointer to inside of a MODEL node.

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

- parsing/network_parser.hpp

6.26 OutputManager Class Reference

Class that outputs formatted resulting data.

```
#include <output_manager.hpp>
```

Public Member Functions

- OutputManager (const ColorStorage &_storage, const ColoringAnalyzer &_analyzer, const SplitManager &_split_manager, WitnessSearcher &_searcher, RobustnessCompute &_robustness)
- void outputSummary (const std::size_t total_count)
- void outputRoundNum ()
- const std::vector< std::string > getCosts (const std::vector< std::size_t > cost_vals) const
- void outputRound () const

6.26.1 Constructor & Destructor Documentation

6.26.1.1 OutputManager::OutputManager (const ColorStorage & *_storage,* const ColoringAnalyzer & *_analyzer,* const SplitManager & *_split_manager,* WitnessSearcher & *_searcher,* RobustnessCompute & *_robustness*) [inline]

Simple constructor that only passes the references.

6.26.2 Member Function Documentation

6.26.2.1 const std::vector<std::string> OutputManager::getCosts (const std::vector< std::size_t > *cost_vals*) const [inline]

Recreate vector of cost values into a vector of strings.

6.26.2.2 void OutputManager::outputRound ( ) const [inline]

Display colors synthetized during current round.

6.26.2.3 void OutputManager::outputRoundNum ( ) [inline]

Outputs round number - if there are no data within, then erase the line each round.

6.26.2.4 void OutputManager::outputSummary (const std::size_t *total_count*) [inline]

Output summary after the computation.

Parameters

total_count	number of all feasible colors

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

- synthesis/output_manager.hpp

6.27 OutputStreamer Class Reference

Class that contains methods for standard and special stream output.

```
#include <output_streamer.hpp>
```

Public Types

- typedef const unsigned int **Trait**

Public Member Functions

- bool testTrait (const unsigned int tested, const unsigned int traits) const
- bool isResultInFile () const
- OutputStreamer ()
- ∼OutputStreamer ()
- void createStreamFile (StreamType stream_type, std::string filename)
- void flush ()
- template<class outputType >
  const OutputStreamer & output (StreamType stream_type, const outputType &stream_data, const unsigned int trait_mask=0)
- template<class outputType >
  const OutputStreamer & output (const outputType &stream_data, const unsigned int trait_mask=0) const

Static Public Attributes

- static Trait no_newl = 1

  After last line no newline symbol will be output.
- static Trait important = 2

  Add "-- " before and " --" after the ouptut.
- static Trait rewrite_ln = 4

  Return the cursor and start from the beginning of the line.
- static Trait tab = 8

  Add " " before the output.

6.27.1 Constructor & Destructor Documentation

6.27.1.1 OutputStreamer::OutputStreamer ( ) `[inline]`

Basic constructor - should be used only for the single object shared throught the program.

6.27.1.2 OutputStreamer::∼OutputStreamer ( ) `[inline]`

If some of the streams has been assigned a file, delete that file object.

6.27.2 Member Function Documentation

6.27.2.1 void OutputStreamer::createStreamFile (StreamType *stream_type,* std::string *filename*) `[inline]`

Create a file to which given stream will be redirected.

Parameters

stream_type	enumeration type specifying the type of stream to output to
data	data to output - should be any possible ostream data

6.27.2.2 void OutputStreamer::flush ( ) `[inline]`

Flush all the streams that are in use.

6.27.2.3 bool OutputStreamer::isResultInFile ( ) const `[inline]`

Returns

true if there is a file to output the results

6.27.2.4 template<class outputType > const OutputStreamer& OutputStreamer::output (StreamType *stream_type*, const outputType & *stream_data*, const unsigned int *trait_mask* = 0) [inline]

Output on a specified stream.

Parameters

stream_type	enumeration type specifying the type of stream to output to
data	data to output - should be any possible ostream data
trait_mask	bitmask of traits for output

6.27.2.5 template<class outputType > const OutputStreamer& OutputStreamer::output (const outputType & *stream_data*, const unsigned int *trait_mask* = 0) const [inline]

Overloaded method that uses the same stream as the last ouput.

Parameters

data	data to output - should be any possible ostream data
trait_mask	bitmask of traits for output

6.27.2.6 bool OutputStreamer::testTrait (const unsigned int *tested*, const unsigned int *traits*) const [inline]

Test if given trait is present.

Parameters

tested	number of the tested trait
traits	traits given with the function

Returns

bool if the trait is present

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

- auxiliary/output_streamer.hpp

6.28 ParametrizationsBuilder Class Reference

Class that computes feasible parametrizations for each specie from edge constrains and stores them in a ParametrizationHolder object.

```
#include <parametrizations_builder.hpp>
```

Public Member Functions

- ParametrizationsBuilder (const Model &_model, ParametrizationsHolder &_parametrizations)

  Empty default constructor.
- void buildParametrizations ()

6.28.1 Member Function Documentation

6.28.1.1 void **ParametrizationsBuilder::buildParametrizations ( )** `[inline]`

Entry function of parsing, tests and stores subcolors for all the species.

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

- construction/parametrizations_builder.hpp

6.29 ParametrizationsHolder Class Reference

Stores partial parametrizations (functions of each component).

`#include <parametrizations_holder.hpp>`

Classes

- struct **SpecieColors**

  Holds all the feasible subcolors for single Specie w.r.t. edge constrains.

Public Member Functions

- ParametrizationsHolder ()

  Default empty constructor, needed to create an empty object that will be filled.
- std::size_t getSpecieNum () const
- std::size_t getAllColorsNum (const SpecieID ID) const
- std::size_t getColorsNum (const SpecieID ID) const
- std::size_t getSpaceSize () const
- const std::vector< std::size_t > & getColor (const SpecieID ID, const ColorNum color_num) const
- const std::vector< std::size_t > getTargetVals (const SpecieID ID, const std::size_t regul_num) const
- const std::string createColorString (ColorNum number) const
- const std::vector< ColorNum > getSpecieVals (ColorNum number) const

Friends

- class **ParametrizationsBuilder**

6.29.1 Detailed Description

This class stores feasible subcolors for each specie are stored with that specie (a vector of all the possibilities for parametrization for this specie).

Attention

Subcolor means partial parametrizatrization $\sim$ full parametrization of a single specie.

6.29.2 Member Function Documentation

6.29.2.1 const std::string **ParametrizationsHolder::createColorString** (ColorNum *number*) const [inline]

This function creates a string containing a parametrization from its ordinal number. The string is in the form [specie_1_context_1, specie_2_context_2,...,specie_m_context_n].

Parameters

number	ordinal number of the parametrization to be converted

Returns

string representation of given parametrisation

6.29.2.2 std::size_t **ParametrizationsHolder::getAllColorsNum** (const SpecieID *ID*) const [inline]

Parameters

ID	ID of the specie to get the number from

Returns

total number of subcolors this specie could have (all regulatory contexts' combinations)

6.29.2.3 const std::vector<std::size_t>& **ParametrizationsHolder::getColor** (const SpecieID *ID,* const ColorNum *color_num*) const [inline]

Parameters

ID	ID of the specie the requested subcolor belongs to
color_num	ordinal number of the requested subcolor

Returns

requested subcolor from the vector of subcolors of given specie

6.29.2.4 std::size_t ParametrizationsHolder::getColorsNum (const SpecieID *ID*) const [inline]

Parameters

ID	ID of the specie to get the number from

Returns

total number of subcolors this specie has (allowed regulatory contexts' combinations)

6.29.2.5 std::size_t ParametrizationsHolder::getSpaceSize ( ) const [inline]

Returns

size of the parameter space used in the computation

6.29.2.6 std::size_t ParametrizationsHolder::getSpecieNum ( ) const [inline]

Returns

total number of species

6.29.2.7 const std::vector<ColorNum> ParametrizationsHolder::getSpecieVals (ColorNum *number*) const [inline]

This function takes ordinal number of a parametrization and computes ordinal number of partial parametrizations it is built from.

Parameters

number	ordinal number of the parametrization to be converted

Returns

ordinal numbers of partial parametrizations in a vector indexed by IDs of the species

6.29.2.8 const std::vector<std::size_t> ParametrizationsHolder::getTargetVals (const SpecieID *ID*, const std::size_t *regul_num*) const [inline]

This function returns a vector containing target value for a given regulatory contexts for ALL the contexts allowed (in lexicographical order).

Parameters

ID	ID of the specie that is regulated
regul_num	ordinal number of the regulatory context (in a lexicographical order)

Returns

vector with a target value for a given specie and regulatory context for each subcolor (parametrization of the single specie)

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

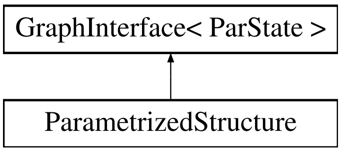
- construction/parametrizations_holder.hpp

6.30 ParametrizedStructure Class Reference

Complete Kripke structure with only possible transitions containing encoded kinetic functions.

`#include <parametrized_structure.hpp>`

Inheritance diagram for ParametrizedStructure:



Public Member Functions

- ParametrizedStructure ()

  Default empty constructor.
- const Levels & getStateLevels (const StateID ID) const
- std::size_t getStepSize (const StateID ID, const std::size_t transtion_num) const
- const std::vector< bool > & getTransitive (const StateID ID, const std::size_t transtion_num) const

Friends

- class **ParametrizedStructureBuilder**

6.30.1 Detailed Description

ParametrizedStructure stores states of the Kripke structure created from the model together with labelled transitions. Each transition contains a function that causes it with explicit enumeration of values from the function that are transitive. To easily search for the values in the parameter bitmask, step_size of the function is added

- that is the value saying how many bits of mask share the the same value for the function. ParametrizedStructure data can be set only from the ParametrizedStructureBuilder object.

6.30.2 Member Function Documentation

6.30.2.1 const Levels& ParametrizedStructure::getStateLevels (const StateID *ID*) const [inline]

Parameters

ID	ID of the state to get the data from

Returns

species level

6.30.2.2 std::size_t ParametrizedStructure::getStepSize (const StateID *ID,* const std::size_t *transtion_num*) const [inline]

Parameters

ID	ID of the state to get the data from
transition_-num	index of the transition to get the data from

Returns

number of neighbour parameters that share the same value of the function

6.30.2.3 const std::vector<bool>& ParametrizedStructure::getTransitive (const StateID *ID,* const std::size_t *transtion_num*) const [inline]

Parameters

ID	ID of the state to get the data from
transition_-num	index of the transition to get the data from

Returns

target values that are includete in non-transitive parameters that have to be removed

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

- construction/parametrized_structure.hpp

6.31 ParametrizedStructureBuilder Class Reference

Creates a ParametrizedStructure as a composition of a BasicStructure and -ParametrizationsHolder.

```
#include <parametrized_structure_builder.hpp>
```

Public Member Functions

- ParametrizedStructureBuilder (const BasicStructure &_basic_structure, const -LabelingHolder &_regulatory_functions, ParametrizedStructure &_structure)
- void buildStructure ()

6.31.1 Detailed Description

ParametrizedStructureBuilder creates the ParametrizedStructure from the model data. States are read from the basic structure and passed to the parametrized structure, then the transitions are added. Each transition is supplemented with a label - mask of transitive values and the its function ID. This expects semantically correct data from BasicStructure and FunctionsStructure.

6.31.2 Constructor & Destructor Documentation

6.31.2.1 ParametrizedStructureBuilder::ParametrizedStructureBuilder (const BasicStructure & *_basic_structure*, const LabelingHolder & *_regulatory_functions*, ParametrizedStructure & *_structure*) [inline]

Constructor just attaches the references to data holders.

6.31.3 Member Function Documentation

6.31.3.1 void ParametrizedStructureBuilder::buildStructure ( ) [inline]

Create the states from the model and fill the structure with them.

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

- construction/parametrized_structure_builder.hpp

6.32 ParamsetHelper Class Reference

Class with mainly static functions for paramset (integer) handling.

`#include <paramset_helper.hpp>`

Public Member Functions

- Paramset getAll () const
- Paramset getLeftOne (ColorNum size=subset_size) const
- std::vector< Paramset > getSingleMasks (Paramset paramset) const
- Paramset getMaskFromNums (const std::vector< std::size_t > numbers) const
- Paramset flip (const Paramset paramset) const
- Paramset swap (Paramset paramset) const
- Paramset swap (Paramset paramset, const std::size_t shift) const
- int **count** (Paramset n) const
- bool none (Paramset paramset) const
- std::size_t getBitNum (Paramset paramset) const

Static Public Member Functions

- static std::size_t getParamsetSize ()

6.32.1 Detailed Description

Here are methods that provide help when working with subsets of parametrization space.

Attention

Parametrizations in an Paramset are ordered in an ascending order.

6.32.2 Member Function Documentation

6.32.2.1 Paramset ParamsetHelper::flip (const Paramset *paramset*) const `[inline]`

Flips every bit.

Parameters

`in`	*paramset*	paramset to flip bits in

Returns

copy of input with swapped bits.

6.32.2.2 Paramset ParamsetHelper::getAll ( ) const `[inline]`

Returns

paramset with everything set to 1

6.32.2.3 std::size_t ParamsetHelper::getBitNum (Paramset *paramset*) const `[inline]`

Get a number of the first on bit.

Parameters

`in`	*paramset*	bitmask that is required to have just one bit on

Returns

position of the bit in the mask (from the left)

6.32.2.4 Paramset ParamsetHelper::getLeftOne (ColorNum *size* = `subset_size`) const `[inline]`

Returns

paramset that holds value of the binary form 10...0 (leftmost parametrization)

6.32.2.5 Paramset ParamsetHelper::getMaskFromNums (const std::vector< std::size_t > *numbers*) const `[inline]`

Return a paramset with on bits corresponding to requested numbers - i.e. for {1,3} we would get 1010...0.

Parameters

in	*vector*	of number in range [1,|paramset|]

Returns

Paramset mask

6.32.2.6 static std::size_t ParamsetHelper::getParamsetSize ( ) [inline, static]

VALUE GETTERS

Returns

number of parametrizations in a single round

6.32.2.7 std::vector<Paramset> ParamsetHelper::getSingleMasks (Paramset *paramset*) const [inline]

TRANSFORMERS Computer a vector of masks of single parametrizations - i.e. 10010 would give {10000,00010}.

Parameters

in	*paramset*	paramset to disassamble

Returns

vector containing a paramset with a single parametrization for each parametrization in input paramset

6.32.2.8 bool ParamsetHelper::none (Paramset *paramset*) const [inline]

Test if none of the parametrizations is present.

Parameters

in	*paramset*	paramset to count bits in

Returns

true if none of the paremters is set

6.32.2.9 Paramset ParamsetHelper::swap (Paramset *paramset*) const [inline]

Swaps paramset within a variable - last become first etc.

Parameters

in	*paramset*	paramset to swap bits in

Returns

copy of input with descending order of paramset

6.32.2.10 Paramset ParamsetHelper::swap (Paramset *paramset,* const std::size_t *shift*) const [inline]

Swaps paramset within a variable - last become first etc.

Parameters

in	*paramset*	paramset to swap bits in
in	*shift*	if there are not all the paramset used, shift back after swapping

Returns

copy of input with descending order of paramset

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

- synthesis/paramset_helper.hpp

6.33 ParsingManager Class Reference

STEP 1 - Class that manages all of the parsing done by the application. Includes parsing of arguments and parsing of models.

```
#include <parsing_manager.hpp>
```

Public Member Functions

- ParsingManager (int argc, char ∗argv[], Model &_model)
- void parse ()

6.33.1 Constructor & Destructor Documentation

6.33.1.1 ParsingManager::ParsingManager (int *argc*, char ∗ *argv[ ]*, **Model** & *_model*) [inline]

Constructor copies arguments from the argv and passes the model object that will store parsed information.

Parameters

argc	passed argc from main()
argv	passed argv from main()
_model	model storing the reference data

6.33.2 Member Function Documentation

6.33.2.1 void **ParsingManager::parse** ( ) [inline]

Main parsing function.

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

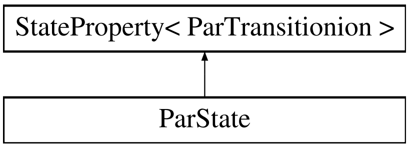
- parsing/parsing_manager.hpp

6.34 ParState Struct Reference

Simple state enriched with transition functions.

`#include <parametrized_structure.hpp>`

Inheritance diagram for ParState:



Public Member Functions

- ParState (const StateID ID, const Levels &_species_level, const std::string &&label)

  Simple filler, assigns values to all the variables.

Public Attributes

- Levels species_level

  Species_level[i] = activation level of specie i.

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

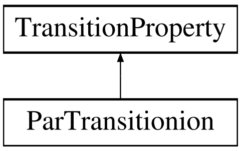
- construction/parametrized_structure.hpp

6.35 ParTransitionion Struct Reference

Storing a single transition to neighbour state together with its transition function.

`#include <parametrized_structure.hpp>`

Inheritance diagram for ParTransitionion:



Public Member Functions

- ParTransitionion (const StateID target_ID, const std::size_t _step_size, std::vector< bool > &&_transitive_values)

  Simple filler, assigns values to all the variables.

Public Attributes

- std::size_t step_size

  How many bits of a parameter space bitset is needed to get from one targe value to another.
- std::vector< bool > transitive_values

  Which values from the original set does not allow a trasition and therefore removes bits from the mask.

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

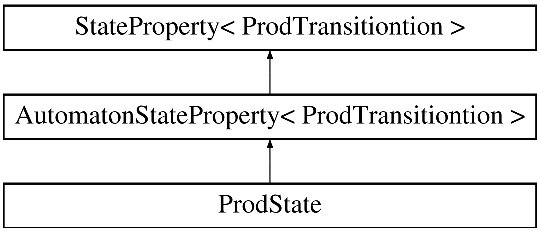
- construction/parametrized_structure.hpp

6.36 ProdState Struct Reference

State of the product - same as the state of parametrized structure but put together with a BA state.

`#include <product_structure.hpp>`

Inheritance diagram for ProdState:


StateProperty< ProdTransitiontion >
AutomatonStateProperty< ProdTransitiontion >
ProdState

Public Member Functions

- ProdState (const StateID ID, const std::string &&label, const StateID _KS_ID, const StateID _BA_ID, const bool initial, const bool final, const Levels &_species_level)

    Simple filler, assigns values to all the variables.

Public Attributes

- StateID KS_ID

    ID of an original KS state this one is built from.
- StateID BA_ID

    ID of an original BA state this one is built from.
- Levels species_level

    species_level[i] = activation level of specie i in this state

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

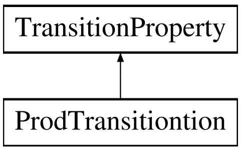
- construction/product_structure.hpp

6.37 ProdTransitiontion Struct Reference

Storing a single transition to a neighbour state together with its transition function.

`#include <product_structure.hpp>`

Inheritance diagram for ProdTransitiontion:



Public Member Functions

- ProdTransitiontion (const StateID target_ID, const std::size_t _step_size, const std::vector< bool > &_transitive_values)

  Simple filler, assigns values to all the variables.

Public Attributes

- std::size_t step_size

  How many bits of a parameter space bitset is needed to get from one targe value to another.
- std::vector< bool > transitive_values

  Which values from the original set does not allow a trasition and therefore removes bits from the mask.

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

- construction/product_structure.hpp

6.38 ProductBuilder Class Reference

Creates a final ProductStructure that is used as a template for the synthesis procedure, as a product of ParametrizedStructure and AutomatonStructure.

```
#include <product_builder.hpp>
```

Public Member Functions

- ProductBuilder (const ParametrizedStructure &_structure, const AutomatonStructure &_automaton, ProductStructure &_product)
- void buildProduct ()

6.38.1 Detailed Description

ProductBuilder creates the an automaton corresponding to the synchronous product of BA and KS.

Attention

> States of product are indexed as (BA_state_count $*$ KS_state_ID + BA_state_ID) - e.g. if 3-state BA state ((1,0)x(1)) would be at position 3$*$1 + 1 = 4.

6.38.2 Constructor & Destructor Documentation

6.38.2.1 ProductBuilder::ProductBuilder (const ParametrizedStructure & *_structure,* const AutomatonStructure & *_automaton,* ProductStructure & *_product*) [inline]

Constructor just attaches the references of data holders.

6.38.3 Member Function Documentation

6.38.3.1 void ProductBuilder::buildProduct ( ) [inline]

Create the the synchronous product of the provided BA and PKS.

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

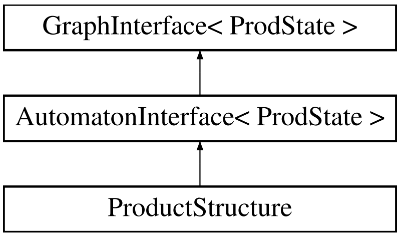
- construction/product_builder.hpp

6.39 ProductStructure Class Reference

Holds a product structure - the one that is used in coloring procedure.

`#include <product_structure.hpp>`

Inheritance diagram for ProductStructure:



Public Member Functions

- ProductStructure (const ParametrizedStructure &_structure, const AutomatonStructure &_automaton)

  Default constructor, only passes the data.
- StateID getProductID (const StateID KS_ID, const StateID BA_ID) const
- StateID getBAID (const StateID ID) const
- StateID getKSID (const StateID ID) const
- const Levels & getStateLevels (const StateID ID) const
- std::size_t getStepSize (const StateID ID, const std::size_t transtion_num) const
- const std::vector< bool > & getTransitive (const StateID ID, const std::size_t transtion_num) const

Friends

- class **ProductBuilder**

6.39.1 Detailed Description

This is the final step of construction - a structure that is acutally used during the computation. For simplicity, it copies data from its predecessors (BA and PKS).

Attention

> States of product are indexed as (BA_state_count $*$ KS_state_ID + BA_state_ID) - e.g. if 3-state BA state ((1,0)x(1)) would be at position 3$*$1 + 1 = 4.

ProductStructure data can be set only from the ProductBuilder object.

6.39.2 Member Function Documentation

6.39.2.1 StateID **ProductStructure::getBAID** (const StateID *ID*) const `[inline]`

Returns

> index of BA state form the product

6.39.2.2 StateID **ProductStructure::getKSID** (const StateID *ID*) const `[inline]`

Returns

> index of BA state form the product

6.39.2.3 StateID **ProductStructure::getProductID** (const StateID *KS_ID*, const StateID *BA_ID*) const `[inline]`

Returns

> index of this combination of states in the product

6.39.2.4 const Levels& **ProductStructure::getStateLevels** (const StateID *ID*) const `[inline]`

Parameters

ID	ID of the state to get the data from

Returns

species level

6.39.2.5 std::size_t **ProductStructure::getStepSize** (const StateID *ID*, const std::size_t *transtion_num*) const [inline]

Parameters

ID	ID of the state to get the data from
transition_-num	index of the transition to get the data from

Returns

number of neighbour parameters that share the same value of the function

6.39.2.6 const std::vector<bool>& **ProductStructure::getTransitive** (const StateID *ID*, const std::size_t *transtion_num*) const [inline]

Parameters

ID	ID of the state to get the data from
transition_-num	index of the transition to get the data from

Returns

target values that are includete in non-transitive parameters that have to be removed

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

- construction/product_structure.hpp

6.40 Model::Regulation Struct Reference

Structure that stores regulation of a specie by another one.

```
#include <model.hpp>
```

Public Member Functions

- **Regulation** (const StateID _source, const std::size_t _threshold, const EdgeConstrain _constrain, const bool _observable)

Public Attributes

- StateID source

  Regulator specie ID.
- std::size_t threshold

  Level of the regulator required for the regulation to be active.
- EdgeConstrain constrain

  What behaviour this regulation must express.
- bool observable

  True if the regulation must be observable.

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

- parsing/model.hpp

6.41 RobustnessCompute Class Reference

Class responsible for computation of robustness values for each acceptable parametrization.

```
#include <robustness_compute.hpp>
```

Classes

- struct **Marking**

  This structure holds values used in the iterative process of robustness computation.

Public Member Functions

- RobustnessCompute (const ConstructionHolder &_holder, const ColorStorage &_storage, const WitnessSearcher &_searcher)
- void compute ()
- const std::vector< std::string > getOutput () const

6.41.1 Constructor & Destructor Documentation

6.41.1.1 RobustnessCompute::RobustnessCompute (const ConstructionHolder & *_holder,* const ColorStorage & *_storage,* const WitnessSearcher & *_searcher*) `[inline]`

Constructor ensures that data objects used within the whole computation process have appropriate size.

6.41.2 Member Function Documentation

6.41.2.1 void RobustnessCompute::compute ( ) [inline]

Function that computes robustness values for each parametrization.

6.41.2.2 const std::vector<std::string> RobustnessCompute::getOutput ( ) const [inline]

Reformes the Robustness values computed to strings. Nothing is produced for parametrizations with 0 robustness.

Returns

a vector of robustness strings

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

- synthesis/robustness_compute.hpp

6.42 SplitManager Class Reference

Class responsible for division of a parametrization space between rounds within a process.

`#include <split_manager.hpp>`

Public Member Functions

- SplitManager (ColorNum _all_colors_count)
- void setStartPositions ()
- bool increaseRound ()
- ColorNum getAllColorsCount () const
- const Range getRoundRange () const
- ColorNum getRoundSize () const
- ColorNum getProcColorsCount () const
- bool lastRound () const
- RoundNum getRoundNum () const
- RoundNum getRoundCount () const
- Paramset createStartingParameters () const

6.42.1 Detailed Description

This class controls splitting of the parameter space both for independent rounds and for distributed synthesis. All data in this class are basic type variables.

6.42.2 Constructor & Destructor Documentation

6.42.2.1 SplitManager::SplitManager (ColorNum *_all_colors_count*) [inline]

Computes splitting for both process (in case of a distributed computation) and its rounds that are of a size of the Parameters data type.

Parameters

_processes-_count	how many processes compute the coloring
process-number	index of this process
_-parameters-_count	complete number of parameters that have to be tested by all the processes

6.42.3 Member Function Documentation

6.42.3.1 Paramset SplitManager::createStartingParameters ( ) const [inline]

Returns

all the parameters of the current round - for the last round, finish has to be cropped

6.42.3.2 ColorNum SplitManager::getAllColorsCount ( ) const [inline]

Returns

total number of parameters for all the processes

6.42.3.3 ColorNum SplitManager::getProcColorsCount ( ) const [inline]

Returns

range with first and one before last parameter to compute for this process

6.42.3.4 RoundNum SplitManager::getRoundCount ( ) const [inline]

Returns

total number of rounds

6.42.3.5 RoundNum SplitManager::getRoundNum ( ) const [inline]

Returns

number of this round

6.42.3.6 const Range SplitManager::getRoundRange ( ) const [inline]

Returns

range with first and one before last parameter to compute this round

6.42.3.7 ColorNum SplitManager::getRoundSize ( ) const [inline]

Returns

number of bits in current round

6.42.3.8 bool SplitManager::increaseRound ( ) [inline]

Increase parameter positions so a new round can be computed.

Returns

true if the increase is possible

6.42.3.9 bool SplitManager::lastRound ( ) const [inline]

Returns

true if this round is not the last

6.42.3.10 void SplitManager::setStartPositions ( ) [inline]

Set values for the first round of computation.

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

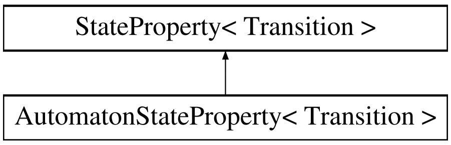
- synthesis/split_manager.hpp

6.43 StateProperty< Transition > Struct Template Reference

This is just a very simple basis for a state of any graph.

`#include <graph_interface.hpp>`

Inheritance diagram for StateProperty< Transition >:


StateProperty< Transition >
AutomatonStateProperty< Transition >

Public Member Functions

- StateProperty (const StateID _ID, const std::string &&_label)

Public Attributes

- StateID ID

  Unique ID of the state.
- std::string label

  Label of the state (usually a number or series of numbers) descibing the state.
- std::vector< Transition > transitions

  Graph or automaton transitions, basically it is an edge with a label.

template< typename Transition > struct StateProperty< Transition >

6.43.1 Constructor & Destructor Documentation

6.43.1.1 template< typename Transition > StateProperty< Transition >::StateProperty (const StateID _ID, const std::string && _label) [inline]

Basic constructor that fills in the ID and label.

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

- construction/graph_interface.hpp

6.44 SynthesisManager Class Reference

STEP 3 - Control class for the computation.

```
#include <synthesis_manager.hpp>
```

Public Member Functions

- SynthesisManager (const ConstructionHolder &_holder)
- void doSynthesis ()

6.44.1 Detailed Description

Manager of the synthesis procedure - takes the reference data constructed during previous steps and computes and executes the synthesis. Synthesis is done in three steps:

1. preparation: empties data and starts a new round.

synthesis: computes the coloring, stored in the storage object and adds data to coloring analyzer if needed.

1. conclusion: stores additional data and outputs

6.44.2 Constructor & Destructor Documentation

6.44.2.1 SynthesisManager::SynthesisManager (const **ConstructionHolder** & *_holder*) [inline]

Constructor builds all the data objects that are used within.

6.44.3 Member Function Documentation

6.44.3.1 void **SynthesisManager::doSynthesis** ( ) [inline]

Main synthesis function that iterates through all the rounds of the synthesis.

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

- synthesis/synthesis_manager.hpp

6.45 TimeManager Class Reference

Class that allows to multiple clock for run-time measurement.

`#include <time_manager.hpp>`

Public Member Functions

- void startClock (const std::string clock_name)
- void ouputClock (const std::string clock_name) const

6.45.1 Member Function Documentation

6.45.1.1 void **TimeManager::ouputClock** (const std::string *clock_name*) const [inline]

Outputs current runtime of the clock.

Parameters

clock_name	name of the clock to output (also appears on the output)

6.45.1.2 void TimeManager::startClock (const std::string *clock_name*) [inline]

Starts a clock with given name and, if it is requsted by user, outputs the info.

Parameters

clock_name	unique ID of the clock that will also be send on the output

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

- auxiliary/time_manager.hpp

6.46 TimeSeriesParser Class Reference

This class parses takes the time series and builds it into a Buchi automaton.

`#include <time_series_parser.hpp>`

Public Member Functions

- TimeSeriesParser (Model &_model)

  Simple constructor, passes references.
- void parse (const rapidxml::xml_node<> *const model_node)

6.46.1 Member Function Documentation

6.46.1.1 void TimeSeriesParser::parse (const rapidxml::xml_node<> *const *model_node*) [inline]

Main parsing function. It expects a pointer to inside of a MODEL node.

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

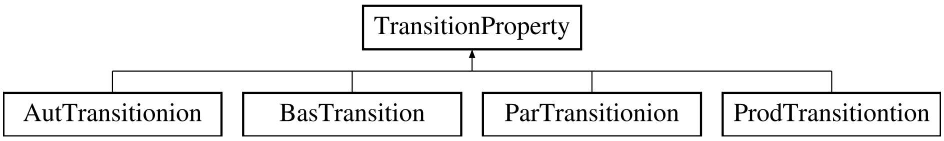
- parsing/time_series_parser.hpp

6.47 TransitionProperty Struct Reference

This is just a very simple basis for a transition in a graph.

`#include <graph_interface.hpp>`

Inheritance diagram for TransitionProperty:



Public Member Functions

- TransitionProperty (StateID _target_ID)

Public Attributes

- StateID target_ID

  Unique ID of the state.

6.47.1 Constructor & Destructor Documentation

6.47.1.1 TransitionProperty::TransitionProperty (StateID *_target_ID*) `[inline]`

Basic constructor fills in the ID.

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

- construction/graph_interface.hpp

6.48 UserOptions Class Reference

Storage of options obtained from execution arguments.

`#include <user_options.hpp>`

Public Member Functions

- UserOptions ()
- bool robustness () const
- bool witnesses () const
- bool analysis () const
- bool longWit () const
- bool verbose () const
- bool stats () const
- bool timeSeries () const
- std::size_t procNum () const

- std::size_t procCount () const
- std::size_t inputMask () const
- std::size_t outputMask () const

Friends

- class **ArgumentParser**
- class **ModelParser**

6.48.1 Detailed Description

Class that stores options provided by the user on the input. Values can be set up only using the ArgumentParser object. Further access to global object user_options is possible only due to constant getters. Only a single object is intended to exist.

6.48.2 Constructor & Destructor Documentation

6.48.2.1 UserOptions::UserOptions ( ) [inline]

Constructor, sets up default values.

6.48.3 Member Function Documentation

6.48.3.1 bool **UserOptions::analysis** ( ) const [inline]

Returns

true if additional analysis will be computed (witnesses/robustness)

6.48.3.2 std::size_t **UserOptions::inputMask** ( ) const [inline]

Returns

true if the input mask was provided

6.48.3.3 bool **UserOptions::longWit** ( ) const [inline]

Returns

true if use_long_witnesses is set (display state levels instead of just a number)

6.48.3.4 std::size_t UserOptions::outputMask ( ) const [inline]

Returns

true if the mask of computation should be printed

6.48.3.5 std::size_t UserOptions::procCount ( ) const [inline]

Returns

total number of processes in distributed computation

6.48.3.6 std::size_t UserOptions::procNum ( ) const [inline]

Returns

number of this process in distributed computation (indexed from 1)

6.48.3.7 bool UserOptions::robustness ( ) const [inline]

Returns

true if compute_robustness (robustness output is requested)

6.48.3.8 bool UserOptions::stats ( ) const [inline]

Returns

true if display_stats is set (displaying statistics of the model)

6.48.3.9 bool UserOptions::timeSeries ( ) const [inline]

Returns

true if property is a time series

6.48.3.10 bool UserOptions::verbose ( ) const [inline]

Returns

true if verbose is set (displaying additional information during computation)

6.48.3.11 bool UserOptions::witnesses ( ) const [inline]

Returns

true if witnesses are to be computed

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

- auxiliary/user_options.hpp

6.49 WitnessSearcher Class Reference

Class for search of transitions belonging to shortest time series paths.

`#include <witness_searcher.hpp>`

Classes

- struct **Marking**

  This structure stores "already tested" paramsets for a state.

Public Member Functions

- WitnessSearcher (const ConstructionHolder &_holder, const ColorStorage &_storage)
- void findWitnesses ()
- const std::vector< std::string > getOutput () const
- const std::vector< std::set < std::pair< StateID, StateID > > > & getTransitions () const
- const std::vector< std::set < StateID > > & getInitials () const

6.49.1 Detailed Description

Class executes a search through the synthetized space in order to find transitions included in shortest paths for every parametrization. Procedure is supposed to be first executed and then it can provide results.

6.49.2 Constructor & Destructor Documentation

6.49.2.1 WitnessSearcher::WitnessSearcher (const ConstructionHolder & *_holder,* const ColorStorage & *_storage*) [inline]

Constructor ensures that data objects used within the whole computation process have appropriate size.

6.49.3 Member Function Documentation

6.49.3.1 void **WitnessSearcher::findWitnesses ( )** [inline]

Function that executes the whole searching process

6.49.3.2 const std::vector<std::set<StateID> >& **WitnessSearcher::getInitials ( )** const [inline]

Returns

a vector of IDs of intial states

6.49.3.3 const std::vector<std::string> **WitnessSearcher::getOutput ( )** const [inline]

Re-formes the transitions computed during the round into strings.

Returns

strings with all transitions for each acceptable parametrization

6.49.3.4 const std::vector<std::set<std::pair<StateID, StateID> > >& **WitnessSearcher::getTransitions ( )** const [inline]

Returns

transitions for each parametrizations in the form (source, target)

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

- synthesis/witness_searcher.hpp