Parsybone

Generated by Doxygen 1.8.1.2

Tue Jul 17 2012 23:01:19

Contents

Chapter 1

Class Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Class Documentation

4.1 ArgumentParser Class Reference

`#include <argument_parser.hpp>`

Public Member Functions

- void parseArguments (UserOptions &user_options, int argc, char ∗argv[])

4.1.1 Detailed Description

Parses arguments on the input and changes switches accordingly. If there is a file on the input, it is created.

Parameters

user_options	parsed data will be saved here
argc	same as at main
argv	same as at main
result_stream	pointer to the stream that will get the output

4.1.2 Member Function Documentation

4.1.2.1 void ArgumentParser::parseArguments (UserOptions & *user_options*, int *argc*, char ∗ *argv[]*) [inline]

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

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

- parsing/argument_parser.hpp

4.2 rapidxml::attribute_iterator< Ch > Class Template Reference

Iterator of child attributes of xml_node.

`#include <rapidxml_iterators.hpp>`

Public Types

- typedef rapidxml::xml_attribute< Ch > **value_type**
- typedef rapidxml::xml_attribute< Ch > & **reference**
- typedef rapidxml::xml_attribute< Ch > * **pointer**
- typedef std::ptrdiff_t **difference_type**
- typedef std::bidirectional_iterator_tag **iterator_category**

Public Member Functions

- **attribute_iterator** (xml_node< Ch > *node)
- reference **operator*** () const
- pointer **operator->** () const
- attribute_iterator & **operator++** ()
- attribute_iterator **operator++** (int)
- attribute_iterator & **operator--** ()
- attribute_iterator **operator--** (int)
- bool **operator==** (const attribute_iterator< Ch > &rhs)
- bool **operator!=** (const attribute_iterator< Ch > &rhs)

4.2.1 Detailed Description

template<class Ch>class rapidxml::attribute_iterator< Ch >

Iterator of child attributes of xml_node.

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

- parsing/rapidxml-1.13/rapidxml_iterators.hpp

4.3 AutomatonBuilder Class Reference

Public Member Functions

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

4.3.1 Constructor & Destructor Documentation

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

Constructor just attaches the references to data holders

4.3.2 Member Function Documentation

4.3.2.1 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:

- reforging/automaton_builder.hpp

4.4 AutomatonInterface Class Reference

Inherits GraphInterface.

Inherited by AutomatonStructure, and ProductStructure.

Public Member Functions

- virtual const bool isFinal (const StateID ID) const =0
- virtual const bool isInitial (const StateID ID) const =0
- virtual const std::vector < StateID > & getFinalStates () const =0
- virtual const std::vector < StateID > & getInitialStates () const =0
- virtual ~AutomatonInterface ()

4.4.1 Constructor & Destructor Documentation

4.4.1.1 virtual AutomatonInterface::~AutomatonInterface ( ) [inline], [virtual]

Virtual destructor.

4.4.2 Member Function Documentation

4.4.2.1 virtual const std::vector<StateID>& AutomatonInterface::getFinalStates ( ) const [inline], [pure virtual]

Get IDs of all states that are marked as final.

Returns

vector of final states' IDs

Implemented in AutomatonStructure, and ProductStructure.

4.4.2.2 virtual const std::vector<StateID>& AutomatonInterface::getInitialStates ( ) const [inline], [pure virtual]

Get IDs of all states that are marked as initial.

Returns

vector of initial states' IDs

Implemented in AutomatonStructure, and ProductStructure.

4.4.2.3 virtual const bool AutomatonInterface::isFinal (const StateID *ID*) const [inline], [pure virtual]

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

Parameters

ID	state to test

Returns

true if the state is final

Implemented in AutomatonStructure, and ProductStructure.

4.4.2.4 virtual const bool AutomatonInterface::isInitial (const StateID *ID*) const [inline], [pure virtual]

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

Parameters

ID	state to test

Returns

true if the state is initial

Implemented in AutomatonStructure, and ProductStructure.

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

- reforging/automaton_interface.hpp

4.5 AutomatonStructure Class Reference

Inherits AutomatonInterface.

Classes

- struct **State**
- struct **Transition**

Public Member Functions

- bool isTransitionFeasible (const std::size_t state_num, const std::size_t transition_num, const Levels &levels) const
- const std::size_t getStateCount () const
- const std::size_t getTransitionCount (const StateID ID) const
- const std::size_t getTargetID (const StateID ID, const std::size_t transition_num) const
- const std::string getString (const StateID ID) const
- virtual const bool isFinal (const StateID ID) const
- virtual const bool isInitial (const StateID ID) const
- virtual const std::vector < StateID > & getFinalStates () const

- virtual const std::vector < StateID > & getInitialStates () const
- const std::vector< std::set < std::size_t > > & getAllowedValues (const StateID ID, const std::size_t transition_num) const

Friends

- class **AutomatonBuilder**

4.5.1 Member Function Documentation

4.5.1.1 const std::vector<std::set<std::size_t> >& AutomatonStructure::getAllowedValues (const StateID *ID,* const std::size_t *transition_num*) const [inline]

Parameters

transition_num	number of transition to get the data from

Returns

ID of the target state of this transition

4.5.1.2 virtual const std::vector<StateID>& AutomatonStructure::getFinalStates ( ) const [inline], [virtual]

Implements AutomatonInterface.

4.5.1.3 virtual const std::vector<StateID>& AutomatonStructure::getInitialStates ( ) const [inline], [virtual]

Only the first state is considered initial.

Implements AutomatonInterface.

4.5.1.4 const std::size_t AutomatonStructure::getStateCount ( ) const [inline], [virtual]

Implements GraphInterface.

4.5.1.5 const std::string AutomatonStructure::getString (const StateID *ID*) const [inline], [virtual]

Return string representing the state in the form: (ID).

Implements GraphInterface.

4.5.1.6 const std::size_t AutomatonStructure::getTargetID (const StateID *ID,* const std::size_t *transition_num*) const [inline], [virtual]

Implements GraphInterface.

4.5.1.7 const std::size_t AutomatonStructure::getTransitionCount (const StateID *ID*) const [inline], [virtual]

Implements GraphInterface.

4.5.1.8 virtual const bool AutomatonStructure::isFinal (const StateID *ID*) const [inline], [virtual]

Implements AutomatonInterface.

4.5.1.9 virtual const bool AutomatonStructure::isInitial (const StateID *ID*) const [inline], [virtual]

Only the first state is considered initial.

Implements AutomatonInterface.

4.5.1.10 bool AutomatonStructure::isTransitionFeasible (const std::size_t *state_num*, const std::size_t *transition_num*, const Levels & *levels*) const [inline]

Parameters

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:

- reforging/automaton_structure.hpp

4.6 BasicStructure Class Reference

Inherits GraphInterface.

Classes

- struct **State**
- struct **Transition**

Public Member Functions

- const std::size_t getStateCount () const
- const std::size_t getTransitionCount (const StateID ID) const
- const std::size_t getTargetID (const StateID ID, const std::size_t trans_number) const
- const std::string getString (const StateID ID) const
- const Levels & getStateLevels (const StateID ID) const
- const std::size_t getSpecieID (const StateID ID, const std::size_t neighbour_index) const
- const Direction getDirection (const StateID ID, const std::size_t neighbour_index) const

Friends

- class **BasicStructureBuilder**

4.6.1 Member Function Documentation

4.6.1.1 const 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

4.6.1.2 const 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

4.6.1.3 const std::size_t BasicStructure::getStateCount ( ) const [inline], [virtual]

Implements GraphInterface.

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

Parameters

ID	ID of the state to get

Returns

levels of the state

4.6.1.5 const std::string BasicStructure::getString (const StateID *ID*) const [inline], [virtual]

Unused, return empty string.

Implements GraphInterface.

4.6.1.6 const std::size_t BasicStructure::getTargetID (const StateID *ID*, const std::size_t *trans_number*) const [inline], [virtual]

Implements GraphInterface.

4.6.1.7 const std::size_t BasicStructure::getTransitionCount (const StateID *ID*) const [inline], [virtual]

Implements GraphInterface.

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

- reforging/basic_structure.hpp

4.7 BasicStructureBuilder Class Reference

Public Member Functions

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

4.7.1 Constructor & Destructor Documentation

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

Constructor initializes basic information from the model

4.7.2 Member Function Documentation

4.7.2.1 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:

- reforging/basic_structure_builder.hpp

4.8 ClassName Class Reference

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

- auxiliary/template.hpp

4.9 ColoringAnalyzer Class Reference

Public Member Functions

- ColoringAnalyzer (const ProductStructure &_product)
- void strartNewRound (const Range &round_range)
- void **display** () const
- void storeResults (const Coloring &results)
- const Parameters getUnion () const
- std::vector< std::pair < std::size_t, std::string > > getColors (Parameters result_parameters) const
- std::vector< std::pair < std::size_t, std::string > > getColors () const

4.9.1 Constructor & Destructor Documentation

4.9.1.1 ColoringAnalyzer::ColoringAnalyzer (const ProductStructure & *_product*) [inline]

Get reference data and create final states that will hold all the computed data

4.9.2 Member Function Documentation

4.9.2.1 std::vector<std::pair<std::size_t, std::string> > ColoringAnalyzer::getColors (Parameters *result_parameters*) const [inline]

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

Parameters

result_parameters	parameters to use

Returns

vector of masks and strings of feasible colors

4.9.2.2 std::vector<std::pair<std::size_t, std::string> > ColoringAnalyzer::getColors ( ) 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

4.9.2.3 const Parameters ColoringAnalyzer::getUnion ( ) const [inline]

Compute merge of all final colors creating a coloring with all feasible colors

Returns

all feasible colors in this round

4.9.2.4 void ColoringAnalyzer::storeResults (const Coloring & *results*) [inline]

Store requested results for a give state of product

4.9.2.5 void ColoringAnalyzer::strartNewRound (const Range & *round_range*) [inline]

Iterates color until it responds to the first parameter of this round

Parameters

round_range	first and one behind last parameter of this round

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

- results/coloring_analyzer.hpp

4.10 ColoringParser Class Reference

Public Member Functions

- ColoringParser ()
- void openFile (const std::string filename)
- void createOutput (const std::string filename)
- void parseMask ()
- void outputComputed (const Parameters parameters)
- const bool input () const
- const bool output () const
- const std::vector< Parameters > & getColors () const
- const std::size_t getColorsCount ()

4.10.1 Constructor & Destructor Documentation

4.10.1.1 ColoringParser::ColoringParser ( ) [inline]

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

4.10.2 Member Function Documentation

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

Create a file to output bitmasks to.

Parameters

filename	path to the file to read from

4.10.2.2 const std::vector<Parameters>& ColoringParser::getColors ( ) const [inline]

Returns

masks for all colors that can be used

4.10.2.3 const std::size_t ColoringParser::getColorsCount ( ) [inline]

Returns

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

4.10.2.4 const bool ColoringParser::input ( ) const [inline]

Returns

true if the mask was provided on the input

4.10.2.5 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

4.10.2.6 const bool ColoringParser::output ( ) const [inline]

Returns

true if the mask is requested on the output

4.10.2.7 void ColoringParser::outputComputed (const Parameters *parameters*) [inline]

Send computed data for this round on the ouput

Parameters

parameters	bitmask of computed feasible colors

4.10.2.8 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

4.11 ColorStorage Class Reference

Classes

- struct **State**

Public Member Functions

- void reset ()
- void addFrom (const ColorStorage &other)
- bool update (const Parameters parameters, const StateID ID)
- bool **soft_update** (const Parameters parameters, const StateID ID)
- bool update (const StateID source_ID, const Parameters parameters, const StateID target_ID)
- const Parameters & getColor (const StateID ID) const
- const std::vector< Coloring > getColor (const std::vector< StateID > &states) const
- std::set< StateID > **getColored** () const
- const Neighbours getNeighbours (const StateID ID, const bool successors, const Parameters color_mask=~0) const
- const std::vector< Parameters > getMarking (const StateID ID, const bool successors) const

Friends

- class **ProductBuilder**

4.11.1 Member Function Documentation

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

Add all values from one coloring structure to another

Parameters

other	structure to copy from

4.11.1.2 const Parameters& ColorStorage::getColor (const StateID *ID*) const [inline]

Parameters

ID	index of the state to ask for parameters

Returns

parameters assigned to the state

4.11.1.3 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

4.11.1.4 const std::vector<Parameters> ColorStorage::getMarking (const StateID *ID,* const bool *successors*) 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

Returns

neigbours for given state and their color

4.11.1.5 const Neighbours ColorStorage::getNeighbours (const StateID *ID,* const bool *successors,* const Parameters *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	bitmask for a given color, if it is not specified, all colors are required

Returns

neigbours for given state

4.11.1.6 void ColorStorage::reset ( ) [inline]

Sets all values for all the states to zero

4.11.1.7 bool ColorStorage::update (const Parameters *parameters*, const StateID *ID*) [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

4.11.1.8 bool ColorStorage::update (const StateID *source_ID*, const Parameters *parameters*, const StateID *target_ID*) [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:

- reforging/color_storage.hpp

4.12 ConstrainsParser Class Reference

Classes

- struct **SpecieColors**

Public Member Functions

- **ConstrainsParser** (const Model &_model)
- void parseConstrains ()
- const std::size_t getSpecieNum () const
- const std::size_t getAllColorsNum (const SpecieID ID) const
- const std::size_t getColorsNum (const SpecieID ID) const
- const std::vector< std::size_t > & getColor (const SpecieID ID, const std::size_t color_num) const
- const std::vector< std::size_t > getTargetVals (const SpecieID ID, const std::size_t regul_num) const

4.12.1 Member Function Documentation

4.12.1.1 const std::size_t ConstrainsParser::getAllColorsNum (const SpecieID *ID*) const [inline]

Returns

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

4.12.1.2 const std::vector<std::size_t>& ConstrainsParser::getColor (const SpecieID *ID,* const std::size_t *color_num*) const [inline]

Returns

requested subcolor from the vector of subcolors of given specie

4.12.1.3 const std::size_t ConstrainsParser::getColorsNum (const SpecieID *ID*) const [inline]

Returns

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

4.12.1.4 const std::size_t ConstrainsParser::getSpecieNum ( ) const [inline]

Returns

total number of species

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

Returns

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

4.12.1.6 void ConstrainsParser::parseConstrains ( ) [inline]

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

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

- parsing/constrains_parser.hpp

4.13 rapidxml::file< Ch > Class Template Reference

Represents data loaded from a file.

`#include <rapidxml_utils.hpp>`

Public Member Functions

- file (const char *filename)
- file (std::basic_istream< Ch > &stream)
- Ch * data ()
- const Ch * data () const
- std::size_t size () const

4.13.1 Detailed Description

template<class Ch = char>class rapidxml::file< Ch >

Represents data loaded from a file.

4.13.2 Constructor & Destructor Documentation

4.13.2.1 template<class Ch = char> rapidxml::file< Ch >::file (const char * *filename*) [inline]

Loads file into the memory. Data will be automatically destroyed by the destructor.

Parameters

filename	Filename to load.

4.13.2.2 template<class Ch = char> rapidxml::file< Ch >::file (std::basic_istream< Ch > & *stream*) [inline]

Loads file into the memory. Data will be automatically destroyed by the destructor

Parameters

stream	Stream to load from

4.13.3 Member Function Documentation

4.13.3.1 template<class Ch = char> Ch* rapidxml::file< Ch >::data ( ) [inline]

Gets file data.

Returns

Pointer to data of file.

4.13.3.2 template<class Ch = char> const Ch* rapidxml::file< Ch >::data ( ) const [inline]

Gets file data.

Returns

Pointer to data of file.

4.13.3.3 template<class Ch = char> std::size_t rapidxml::file< Ch >::size ( ) const [inline]

Gets file data size.

Returns

Size of file data, in characters.

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

- parsing/rapidxml-1.13/rapidxml_utils.hpp

4.14 FunctionsBuilder Class Reference

Public Member Functions

- FunctionsBuilder (const Model &_model, const ConstrainsParser &_constrains, FunctionsStructure &_functions_structure)
- void buildFunctions ()

4.14.1 Constructor & Destructor Documentation

4.14.1.1 FunctionsBuilder::FunctionsBuilder (const Model & *_model,* const ConstrainsParser & *_constrains,* FunctionsStructure & *_functions_structure*) [inline]

Constructor just attaches the references to data holders

4.14.2 Member Function Documentation

4.14.2.1 void FunctionsBuilder::buildFunctions ( ) [inline]

For each specie recreate all its regulatory functions

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

- reforging/functions_builder.hpp

4.15 FunctionsStructure Class Reference

Classes

- struct **RegulatoryFunction**
- struct **Specie**

Public Member Functions

- const std::size_t getParametersCount () const
- 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
- const std::size_t getRegulationsCount (const std::size_t ID) const
- 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 **FunctionsBuilder**

4.15.1 Member Function Documentation

4.15.1.1 const std::size_t FunctionsStructure::getParametersCount ( ) const [inline]

Returns

size of the parameter space

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

Returns

values this function can possibly regulate to

4.15.1.3 const std::size_t FunctionsStructure::getRegulationsCount (const std::size_t *ID*) const [inline]

Returns

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

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

Returns

IDs of all the species that regulate this specie

4.15.1.5 const std::vector<std::vector<std::size_t> >& FunctionsStructure::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

4.15.1.6 const std::string& FunctionsStructure::getSpecieName (const std::size_t *ID*) const [inline]

Returns

name of the specie with given ID

4.15.1.7 const std::size_t FunctionsStructure::getSpeciesCount ( ) const [inline]

Returns

number of the species

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

Returns

all the values the specie can occur in

4.15.1.9 const std::size_t FunctionsStructure::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:

- reforging/functions_structure.hpp

4.16 GraphInterface Class Reference

Inherited by AutomatonInterface, BasicStructure, and ParametrizedStructure.

Public Member Functions

- virtual const std::size_t getStateCount () const =0
- virtual const std::size_t getTransitionCount (const StateID ID) const =0
- virtual const StateID getTargetID (const StateID ID, const std::size_t transition_number) const =0
- virtual const std::string getString (const std::size_t StateID) const =0
- virtual ~GraphInterface ()

4.16.1 Constructor & Destructor Documentation

4.16.1.1 virtual GraphInterface::~GraphInterface ( ) [inline], [virtual]

Virtual destructor.

4.16.2 Member Function Documentation

4.16.2.1 virtual const std::size_t GraphInterface::getStateCount ( ) const [inline], [pure virtual]

Obtains number of states of the graph.

Returns

integer with size of the graph

Implemented in AutomatonStructure, ProductStructure, ParametrizedStructure, and BasicStructure.

4.16.2.2 virtual const std::string GraphInterface::getString (const std::size_t *StateID*) const [inline], [pure virtual]

Returns given state as a string.

Parameters

ID	ID of the state to turn into the string

Returns

given state as a string

Implemented in AutomatonStructure, ProductStructure, ParametrizedStructure, and BasicStructure.

4.16.2.3 virtual const StateID GraphInterface::getTargetID (const StateID *ID,* const std::size_t *transition_number*) const [inline],[pure virtual]

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

Implemented in AutomatonStructure, ProductStructure, ParametrizedStructure, and BasicStructure.

4.16.2.4 virtual const std::size_t GraphInterface::getTransitionCount (const StateID *ID*) const [inline],[pure virtual]

Obtains number of transitions for given state.

Parameters

ID	ID of the state to get the number from

Returns

integer with number of outcoming transitions

Implemented in AutomatonStructure, ProductStructure, ParametrizedStructure, and BasicStructure.

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

- reforging/graph_interface.hpp

4.17 Model::Interaction Struct Reference

Public Member Functions

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

Public Attributes

- StateID **source**
- std::size_t **threshold**
- EdgeConstrain **constrain**
- bool **observable**

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

- parsing/model.hpp

4.18 rapidxml::memory_pool< Ch > Class Template Reference

`#include <rapidxml.hpp>`

Inherited by rapidxml::xml_document< Ch >.

Classes

- struct **header**

Public Member Functions

- memory_pool ()

  Constructs empty pool with default allocator functions.
- ~memory_pool ()
- xml_node< Ch > * allocate_node (node_type type, const Ch *name=0, const Ch *value=0, std::size_t name_size=0, std::size_t value_size=0)
- xml_attribute< Ch > * allocate_attribute (const Ch *name=0, const Ch *value=0, std::size_t name_size=0, std::size_t value_size=0)
- Ch * allocate_string (const Ch *source=0, std::size_t size=0)
- xml_node< Ch > * clone_node (const xml_node< Ch > *source, xml_node< Ch > *result=0)
- void clear ()
- void set_allocator (alloc_func *af, free_func *ff)

4.18.1 Detailed Description

template<class Ch = char>class rapidxml::memory_pool< Ch >

This class is used by the parser to create new nodes and attributes, without overheads of dynamic memory allocation. In most cases, you will not need to use this class directly. However, if you need to create nodes manually or modify names/values of nodes, you are encouraged to use memory_pool of relevant xml_document to allocate the memory. Not only is this faster than allocating them by using `new` operator, but also their lifetime will be tied to the lifetime of document, possibly simplyfing memory management.

Call allocate_node() or allocate_attribute() functions to obtain new nodes or attributes from the pool. You can also call allocate_string() function to allocate strings. Such strings can then be used as names or values of nodes without worrying about their lifetime. Note that there is no `free()` function – all allocations are freed at once when clear() function is called, or when the pool is destroyed.

It is also possible to create a standalone memory_pool, and use it to allocate nodes, whose lifetime will not be tied to any document.

Pool maintains `RAPIDXML_STATIC_POOL_SIZE` bytes of statically allocated memory. Until static memory is exhausted, no dynamic memory allocations are done. When static memory is exhausted, pool allocates additional blocks of memory of size `RAPIDXML_DYNAMIC_POOL_SIZE` each, by using global `new[]` and `delete[]` operators. This behaviour can be changed by setting custom allocation routines. Use set_allocator() function to set them.

Allocations for nodes, attributes and strings are aligned at `RAPIDXML_ALIGNMENT` bytes. This value defaults to the size of pointer on target architecture.

To obtain absolutely top performance from the parser, it is important that all nodes are allocated from a single, contiguous block of memory. Otherwise, cache misses when jumping between two (or more) disjoint blocks of

memory can slow down parsing quite considerably. If required, you can tweak RAPIDXML_STATIC_POOL_SIZE, RAPIDXML_DYNAMIC_POOL_SIZE and RAPIDXML_ALIGNMENT to obtain best wasted memory to performance compromise. To do it, define their values before rapidxml.hpp file is included.

Parameters

Ch	Character type of created nodes.

4.18.2 Constructor & Destructor Documentation

4.18.2.1 template<class Ch = char> rapidxml::memory_pool< Ch >::~memory_pool ( ) [inline]

Destroys pool and frees all the memory. This causes memory occupied by nodes allocated by the pool to be freed. Nodes allocated from the pool are no longer valid.

4.18.3 Member Function Documentation

4.18.3.1 template<class Ch = char> xml_attribute<Ch>* rapidxml::memory_pool< Ch >::allocate_attribute (const Ch * *name* = 0, const Ch * *value* = 0, std::size_t *name_size* = 0, std::size_t *value_size* = 0) [inline]

Allocates a new attribute from the pool, and optionally assigns name and value to it. If the allocation request cannot be accomodated, this function will throw std::bad_alloc. If exceptions are disabled by defining RAPIDXML_NO_EXCEPTIONS, this function will call rapidxml::parse_error_handler() function.

Parameters

name	Name to assign to the attribute, or 0 to assign no name.
value	Value to assign to the attribute, or 0 to assign no value.
name_size	Size of name to assign, or 0 to automatically calculate size from name string.
value_size	Size of value to assign, or 0 to automatically calculate size from value string.

Returns

Pointer to allocated attribute. This pointer will never be NULL.

4.18.3.2 template<class Ch = char> xml_node<Ch>* rapidxml::memory_pool< Ch >::allocate_node (node_type *type*, const Ch * *name* = 0, const Ch * *value* = 0, std::size_t *name_size* = 0, std::size_t *value_size* = 0) [inline]

Allocates a new node from the pool, and optionally assigns name and value to it. If the allocation request cannot be accomodated, this function will throw std::bad_alloc. If exceptions are disabled by defining RAPIDXML_NO_EXCEPTIONS, this function will call rapidxml::parse_error_handler() function.

Parameters

type	Type of node to create.
name	Name to assign to the node, or 0 to assign no name.
value	Value to assign to the node, or 0 to assign no value.
name_size	Size of name to assign, or 0 to automatically calculate size from name string.
value_size	Size of value to assign, or 0 to automatically calculate size from value string.

Returns

Pointer to allocated node. This pointer will never be NULL.

4.18.3.3 template<class Ch = char> Ch* rapidxml::memory_pool< Ch >::allocate_string (const Ch * *source* = 0, std::size_t *size* = 0) [inline]

Allocates a char array of given size from the pool, and optionally copies a given string to it. If the allocation request cannot be accomodated, this function will throw `std::bad_alloc`. If exceptions are disabled by defining RAPIDXML_NO_EXCEPTIONS, this function will call rapidxml::parse_error_handler() function.

Parameters

source	String to initialize the allocated memory with, or 0 to not initialize it.
size	Number of characters to allocate, or zero to calculate it automatically from source string length; if size is 0, source string must be specified and null terminated.

Returns

Pointer to allocated char array. This pointer will never be NULL.

4.18.3.4 template<class Ch = char> void rapidxml::memory_pool< Ch >::clear ( ) [inline]

Clears the pool. This causes memory occupied by nodes allocated by the pool to be freed. Any nodes or strings allocated from the pool will no longer be valid.

Reimplemented in rapidxml::xml_document< Ch >.

4.18.3.5 template<class Ch = char> xml_node<Ch>* rapidxml::memory_pool< Ch >::clone_node (const xml_node< Ch > * *source*, xml_node< Ch > * *result* = 0) [inline]

Clones an xml_node and its hierarchy of child nodes and attributes. Nodes and attributes are allocated from this memory pool. Names and values are not cloned, they are shared between the clone and the source. Result node can be optionally specified as a second parameter, in which case its contents will be replaced with cloned source node. This is useful when you want to clone entire document.

Parameters

source	Node to clone.
result	Node to put results in, or 0 to automatically allocate result node

Returns

Pointer to cloned node. This pointer will never be NULL.

4.18.3.6 template<class Ch = char> void rapidxml::memory_pool< Ch >::set_allocator (alloc_func * *af*, free_func * *ff*) [inline]

Sets or resets the user-defined memory allocation functions for the pool. This can only be called when no memory is allocated from the pool yet, otherwise results are undefined. Allocation function must not return invalid pointer on failure. It should either throw, stop the program, or use `longjmp()` function to pass control to other place of program. If it returns invalid pointer, results are undefined.

User defined allocation functions must have the following forms:

```
void *allocate(std::size_t size);
void free(void *pointer);
```

Parameters

af	Allocation function, or 0 to restore default function
ff	Free function, or 0 to restore default function

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

- parsing/rapidxml-1.13/rapidxml.hpp

4.19 Model Class Reference

Classes

- struct **AdditionalInformation**
- struct **BuchiAutomatonState**
- struct Interaction
- struct **ModelSpecie**

Public Types

- typedef std::pair< std::vector < bool >, int > **Regulation**
- typedef std::pair< StateID, std::string > **Egde**

Public Member Functions

- const std::size_t getSpeciesCount () const
- const std::size_t getStateCount () const
- const int findID (const std::string name) const
- const std::string & getName (const std::size_t ID) const
- const std::size_t getMin (const std::size_t ID) const
- const std::size_t getMax (const std::size_t ID) const
- const std::vector< Interaction > & getInteractions (const std::size_t ID) const
- const std::vector< Regulation > & getRegulations (const std::size_t ID) const
- const bool isFinal (const std::size_t ID) const
- const std::vector< Egde > & getEdges (const std::size_t ID) const

Friends

- class **ModelParser**

4.19.1 Member Function Documentation

4.19.1.1 const int Model::findID (const std::string *name*) const [inline]

Returns

ID of the specie with the specified name if there is such, otherwise -1

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

Returns

edges of the state

4.19.1.3 const std::vector<Interaction>& Model::getInteractions (const std::size_t *ID*) const [inline]

Returns

interactions of the specie

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

Returns

maximal value of the specie

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

Returns

minimal value of the specie (always 0)

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

Returns

name of the specie

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

Returns

regulations of the specie

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

Returns

number of the species

4.19.1.9 const std::size_t Model::getStateCount ( ) const [inline]

Returns

number of the states

4.19.1.10 const 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

4.20 ModelChecker Class Reference

Public Member Functions

- ModelChecker (const ProductStructure &_product, ColorStorage &_storage)
- const std::vector< std::size_t > startColoring (const StateID ID, const Parameters parameters, const Range &_range, const WitnessUse _witness_use=none_wit)
- const std::vector< std::size_t > startColoring (const Parameters parameters, const std::set< StateID > &_updates, const Range &_range, const WitnessUse _witness_use=none_wit)

4.20.1 Constructor & Destructor Documentation

4.20.1.1 ModelChecker::ModelChecker (const ProductStructure & *_product,* ColorStorage & *_storage*) [inline]

Constructor, passes the data

4.20.2 Member Function Documentation

4.20.2.1 const std::vector<std::size_t> ModelChecker::startColoring (const StateID *ID,* const Parameters *parameters,* const Range & *_range,* const WitnessUse *_witness_use* = none_wit) [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
_witness_use	how to manage witnesses in this coloring round

4.20.2.2 const std::vector<std::size_t> ModelChecker::startColoring (const Parameters *parameters,* const std::set< StateID > & *_updates,* const Range & *_range,* const WitnessUse *_witness_use* = none_wit) [inline]

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

Parameters

_updates	states that are will be scheduled for an update in this round
_range	range of parameters for this coloring round
_witness_use	how to manage witnesses in this coloring round

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

- coloring/model_checker.hpp

4.21 ModelParser Class Reference

Public Member Functions

- ModelParser (Model &_model)
- void parseInput ()

4.21.1 Constructor & Destructor Documentation

4.21.1.1 ModelParser::ModelParser (Model & *_model*) [inline]

Constructor has to provide references to an input stream to read from and model object to store parsed information.

4.21.2 Member Function Documentation

4.21.2.1 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.

Returns

version of the parsed file.

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

- parsing/model_parser.hpp

4.22 rapidxml::node_iterator< Ch > Class Template Reference

Iterator of child nodes of xml_node.

`#include <rapidxml_iterators.hpp>`

Public Types

- typedef rapidxml::xml_node< Ch > **value_type**
- typedef rapidxml::xml_node< Ch > & **reference**
- typedef rapidxml::xml_node< Ch > * **pointer**
- typedef std::ptrdiff_t **difference_type**
- typedef std::bidirectional_iterator_tag **iterator_category**

Public Member Functions

- **node_iterator** (xml_node< Ch > *node)
- reference **operator*** () const
- pointer **operator->** () const
- node_iterator & **operator++** ()
- node_iterator **operator++** (int)
- node_iterator & **operator--** ()
- node_iterator **operator--** (int)
- bool **operator==** (const node_iterator< Ch > &rhs)
- bool **operator!=** (const node_iterator< Ch > &rhs)

4.22.1 Detailed Description

template<class Ch>class rapidxml::node_iterator< Ch >

Iterator of child nodes of xml_node.

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

- parsing/rapidxml-1.13/rapidxml_iterators.hpp

4.23 OutputManager Class Reference

Public Member Functions

- **OutputManager** (const ColoringAnalyzer &_analyzer, const ProductStructure &_product, const SplitManager &_split_manager, WitnessSearcher &_searcher)
- void outputSummary (const std::size_t total_count)
- void outputRoundNum ()
- void outputRound (const std::vector< std::size_t > &BFS_reach) const

4.23.1 Member Function Documentation

4.23.1.1 void OutputManager::outputRound (const std::vector< std::size_t > & *BFS_reach*) const [inline]

Display colors synthetized during current round

4.23.1.2 void OutputManager::outputRoundNum ( ) [inline]

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

4.23.1.3 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:

- results/output_manager.hpp

4.24 OutputStreamer Class Reference

Public Types

- typedef const unsigned int **Trait**

Public Member Functions

- bool testTrait (const unsigned int tested, const unsigned int traits) const

- 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
- static Trait **important** = 2
- static Trait **rewrite_ln** = 4

4.24.1 Constructor & Destructor Documentation

4.24.1.1 OutputStreamer::OutputStreamer ( ) [inline]

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

4.24.1.2 OutputStreamer::~OutputStreamer ( ) [inline]

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

4.24.2 Member Function Documentation

4.24.2.1 void OutputStreamer::createStreamFile (StreamType *stream_type*, std::string *filename*) [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

4.24.2.2 const bool OutputStreamer::isResultInFile ( ) const [inline]

Returns

true if there is a file to output the results

4.24.2.3 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

4.24.2.4 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

4.24.2.5 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

4.25 ParametrizedStructure Class Reference

Inherits GraphInterface.

Classes

- struct **State**
- struct **Transition**

Public Member Functions

- const std::size_t getStateCount () const
- const std::size_t getTransitionCount (const StateID ID) const
- const std::size_t getTargetID (const StateID ID, const std::size_t transtion_num) const
- const std::string getString (const StateID ID) const
- const Levels & getStateLevels (const StateID ID) const
- 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**

4.25.1 Member Function Documentation

4.25.1.1 const std::size_t ParametrizedStructure::getStateCount ( ) const [inline], [virtual]

Implements GraphInterface.

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

Parameters

ID	ID of the state to get the data from

Returns

species level

4.25.1.3 const 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

4.25.1.4 const std::string ParametrizedStructure::getString (const StateID *ID*) const [inline], [virtual]

Return string representing given state in the form (specie1_val, specie2_val, ...)

Implements GraphInterface.

4.25.1.5 const std::size_t ParametrizedStructure::getTargetID (const StateID *ID*, const std::size_t *transtion_num*) const [inline], [virtual]

Implements GraphInterface.

4.25.1.6 const std::size_t ParametrizedStructure::getTransitionCount (const StateID *ID*) const [inline], [virtual]

Implements GraphInterface.

4.25.1.7 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:

- reforging/parametrized_structure.hpp

4.26 ParametrizedStructureBuilder Class Reference

Public Member Functions

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

4.26.1 Constructor & Destructor Documentation

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

Constructor just attaches the references to data holders

4.26.2 Member Function Documentation

4.26.2.1 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:

- reforging/parametrized_structure_builder.hpp

4.27 rapidxml::parse_error Class Reference

`#include <rapidxml.hpp>`

Public Member Functions

- parse_error (const char *what, void *where)

  Constructs parse error.
- virtual const char * what () const throw ()
- template<class Ch >
  Ch * where () const

4.27.1 Detailed Description

Parse error exception. This exception is thrown by the parser when an error occurs. Use what() function to get human-readable error message. Use where() function to get a pointer to position within source text where error was detected.

If throwing exceptions by the parser is undesirable, it can be disabled by defining RAPIDXML_NO_EXCEPTIONS macro before rapidxml.hpp is included. This will cause the parser to call rapidxml::parse_error_handler() function instead of throwing an exception. This function must be defined by the user.

This class derives from `std::exception` class.

4.27.2 Member Function Documentation

4.27.2.1 virtual const char* rapidxml::parse_error::what ( ) const throw () [inline], [virtual]

Gets human readable description of error.

Returns

Pointer to null terminated description of the error.

4.27.2.2 template<class Ch > Ch* rapidxml::parse_error::where ( ) const [inline]

Gets pointer to character data where error happened. Ch should be the same as char type of xml_document that produced the error.

Returns

Pointer to location within the parsed string where error occured.

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

- parsing/rapidxml-1.13/rapidxml.hpp

4.28 PerColorStorage Class Reference

Classes

- struct **ColorData**
- struct **State**

Public Member Functions

- **PerColorStorage** (const ColoringAnalyzer &_analyzer, const ColorStorage &_storage, const ProductStructure &_product)
- const Neighbours & getNeighbours (const StateID ID, const bool successors, std::size_t number) const

4.28.1 Member Function Documentation

4.28.1.1 const Neighbours& PerColorStorage::getNeighbours (const StateID *ID*, const bool *successors*, std::size_t *number*) 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
number	ordinal number of stored coloring

Returns

neigbours for given state

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

- results/per_color_storage.hpp

4.29 ProductBuilder Class Reference

Public Member Functions

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

4.29.1 Constructor & Destructor Documentation

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

Constructor just attaches the references to data holders

4.29.2 Member Function Documentation

4.29.2.1 void ProductBuilder::buildProduct ( ) [inline]

Create the the product from BA and KS together.

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

- reforging/product_builder.hpp

4.30 ProductStructure Class Reference

Inherits AutomatonInterface.

Classes

- struct **State**
- struct **Transition**

Public Member Functions

- **ProductStructure** (const FunctionsStructure &_functions, const ConstrainsParser &_constrains, const ParametrizedStructure &_structure, const AutomatonStructure &_automaton)
- const std::size_t getStateCount () const
- const std::size_t getTransitionCount (const StateID ID) const
- const std::size_t getTargetID (const StateID ID, const std::size_t trans_number) const
- const std::string getString (const StateID ID) const
- virtual const bool isFinal (const StateID ID) const

- virtual const bool isInitial (const StateID ID) const
- virtual const std::vector < StateID > & getFinalStates () const
- virtual const std::vector < StateID > & getInitialStates () const
- const StateID getProductID (const StateID KS_ID, const StateID BA_ID) const
- const StateID getBAID (const StateID ID) const
- const StateID getKSID (const StateID ID) const
- const ParametrizedStructure & getKS () const
- const AutomatonStructure & getBA () const
- const FunctionsStructure & getFunc () const
- const ConstrainsParser & getCons () const
- const Levels & getStateLevels (const StateID ID) const
- 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**

4.30.1 Member Function Documentation

4.30.1.1 const AutomatonStructure& ProductStructure::getBA ( ) const [inline]

Returns

constant reference to Buchi automaton stored within the product

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

Returns

index of BA state form the product

4.30.1.3 const ConstrainsParser& ProductStructure::getCons ( ) const [inline]

Returns

constant reference to structure with interactions constrains

4.30.1.4 virtual const std::vector<StateID>& ProductStructure::getFinalStates ( ) const [inline], [virtual]

Implements AutomatonInterface.

4.30.1.5 const FunctionsStructure& ProductStructure::getFunc ( ) const [inline]

Returns

constant reference to structure with regulatory functions

4.30.1.6 virtual const std::vector<StateID>& ProductStructure::getInitialStates ( ) const [inline], [virtual]

Implements AutomatonInterface.

4.30.1.7 const ParametrizedStructure& ProductStructure::getKS ( ) const [inline]

Returns

constant reference to Kripke structure stored within the product

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

Returns

index of BA state form the product

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

Returns

index of this combination of states in the product

4.30.1.10 const std::size_t ProductStructure::getStateCount ( ) const [inline], [virtual]

Implements GraphInterface.

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

Parameters

ID	ID of the state to get the data from

Returns

species level

4.30.1.12 const 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

4.30.1.13 const std::string ProductStructure::getString (const StateID *ID*) const [inline], [virtual]

Create string in the form KSstateBAstate or KSstate based on if user requests BA as well

Implements GraphInterface.

4.30.1.14 const std::size_t ProductStructure::getTargetID (const StateID *ID,* const std::size_t *trans_number*) const [inline], [virtual]

Implements GraphInterface.

4.30.1.15 const std::size_t ProductStructure::getTransitionCount (const StateID *ID*) const [inline], [virtual]

Implements GraphInterface.

4.30.1.16 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

4.30.1.17 virtual const bool ProductStructure::isFinal (const StateID *ID*) const [inline], [virtual]

Implements AutomatonInterface.

4.30.1.18 virtual const bool ProductStructure::isInitial (const StateID *ID*) const [inline], [virtual]

Implements AutomatonInterface.

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

- reforging/product_structure.hpp

4.31 SplitManager Class Reference

Public Member Functions

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

4.31.1 Constructor & Destructor Documentation

4.31.1.1 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

4.31.2 Member Function Documentation

4.31.2.1 Parameters SplitManager::createStartingParameters ( ) const [inline]

Returns

All the parameters of the current round.

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

Returns

total number of parameters for all the processes

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

Returns

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

4.31.2.4 const long long SplitManager::getRoundCount ( ) const [inline]

Returns

total number of rounds

4.31.2.5 const long long SplitManager::getRoundNum ( ) const [inline]

Returns

number of this round

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

Returns

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

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

Returns

number of bits in current round

4.31.2.8 void SplitManager::increaseRound ( ) [inline]

Increase parameter positions so a new round can be computed.

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

Returns

true if this round is not the last

4.31.2.10 void SplitManager::setStartPositions ( ) [inline]

Set values for the first round of computation.

4.31.2.11 const bool SplitManager::valid ( ) const [inline]

Returns

true if current round is valid (this round does not correspond to any paramteres)

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

- coloring/split_manager.hpp

4.32 SynthesisManager Class Reference

Public Member Functions

- SynthesisManager (const ProductStructure &_product, ColorStorage &_storage)
- void doSynthesis ()

4.32.1 Constructor & Destructor Documentation

4.32.1.1 SynthesisManager::SynthesisManager (const ProductStructure & *_product,* ColorStorage & *_storage*) [inline]

Constructor builds all the data objects that are used within

4.32.2 Member Function Documentation

4.32.2.1 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:

- coloring/synthesis_manager.hpp

4.33 TimeManager Class Reference

Public Member Functions

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

4.33.1 Member Function Documentation

4.33.1.1 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)

4.33.1.2 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

4.34 UserOptions Class Reference

Public Member Functions

- UserOptions ()
- const bool coloring () const
- const WitnessUse witnesses () const
- const bool BA () const
- const bool verbose () const
- const bool stats () const
- const bool negation () const
- const bool timeSerie () const
- const std::size_t procNum () const
- const std::size_t procCount () const
- const bool robustness () const
- const bool displayWintess () const

Friends

- class **ArgumentParser**

4.34.1 Constructor & Destructor Documentation

4.34.1.1 UserOptions::UserOptions ( ) [inline]

Constructor, sets up default values

4.34.2 Member Function Documentation

4.34.2.1 const bool UserOptions::BA ( ) const [inline]

Returns

true if add_BA_to_witness is set (displaying path with BA states as well)

4.34.2.2 const bool UserOptions::coloring ( ) const [inline]

Returns

true if show_coloring is set (displaying each accepting color)

4.34.2.3 const bool UserOptions::displayWintess ( ) const [inline]

Returns

true if display_wintess (witness output is requested)

4.34.2.4 const bool UserOptions::negation ( ) const [inline]

Returns

true if time_serie (checking only reachability)

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

Returns

total number of processes in distributed computation

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

Returns

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

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

Returns

true if compute_robustness (robustness output is requested)

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

Returns

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

4.34.2.9 const bool UserOptions::timeSerie ( ) const [inline]

Returns

true if negative_check (switching feasible for non-feasible)

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

Returns

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

4.34.2.11 const WitnessUse UserOptions::witnesses ( ) const [inline]

Returns

how to manage witnesses

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

- auxiliary/user_options.hpp

4.35 WitnessSearcher Class Reference

Public Member Functions

- WitnessSearcher (const ColoringAnalyzer &_analyzer, const ColorStorage &_storage, const ProductStructure &_product)
- void display (const std::vector< std::size_t > &BFS_reach)

4.35.1 Constructor & Destructor Documentation

4.35.1.1 WitnessSearcher::WitnessSearcher (const ColoringAnalyzer & *_analyzer,* const ColorStorage & *_storage,* const ProductStructure & *_product*) [inline]

Get reference data and create final states that will hold all the computed data

4.35.2 Member Function Documentation

4.35.2.1 void WitnessSearcher::display (const std::vector< std::size_t > & *BFS_reach*) [inline]

Output all witnesses for all colors, might be together with the colors as well.

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

- results/witness_searcher.hpp

4.36 rapidxml::xml_attribute< Ch > Class Template Reference

`#include <rapidxml.hpp>`

Inherits rapidxml::xml_base< Ch >.

Public Member Functions

- xml_attribute ()
- xml_document< Ch > * document () const
- xml_attribute< Ch > * previous_attribute (const Ch *name=0, std::size_t name_size=0, bool case_sensitive=true) const
- xml_attribute< Ch > * next_attribute (const Ch *name=0, std::size_t name_size=0, bool case_sensitive=true) const

Friends

- class **xml_node< Ch >**

Additional Inherited Members

4.36.1 Detailed Description

template<class Ch = char>class rapidxml::xml_attribute< Ch >

Class representing attribute node of XML document. Each attribute has name and value strings, which are available through name() and value() functions (inherited from xml_base). Note that after parse, both name and value of attribute will point to interior of source text used for parsing. Thus, this text must persist in memory for the lifetime of attribute.

Parameters

Ch	Character type to use.

4.36.2 Constructor & Destructor Documentation

4.36.2.1 template<class Ch = char> rapidxml::xml_attribute< Ch >::xml_attribute ( ) [inline]

Constructs an empty attribute with the specified type. Consider using memory_pool of appropriate xml_document if allocating attributes manually.

4.36.3 Member Function Documentation

4.36.3.1 template<class Ch = char> xml_document<Ch>* rapidxml::xml_attribute< Ch >::document ( ) const [inline]

Gets document of which attribute is a child.

Returns

Pointer to document that contains this attribute, or 0 if there is no parent document.

4.36.3.2 template<class Ch = char> xml_attribute<Ch>* rapidxml::xml_attribute< Ch >::next_attribute (const Ch * *name* = 0, std::size_t *name_size* = 0, bool *case_sensitive* = true) const [inline]

Gets next attribute, optionally matching attribute name.

Parameters

name	Name of attribute to find, or 0 to return next attribute regardless of its name; this string doesn't have to be zero-terminated if name_size is non-zero
name_size	Size of name, in characters, or 0 to have size calculated automatically from string
case_sensitive	Should name comparison be case-sensitive; non case-sensitive comparison works properly only for ASCII characters

Returns

Pointer to found attribute, or 0 if not found.

4.36.3.3 template<class Ch = char> xml_attribute<Ch>* rapidxml::xml_attribute< Ch >::previous_attribute (const Ch * *name* = 0, std::size_t *name_size* = 0, bool *case_sensitive* = true) const [inline]

Gets previous attribute, optionally matching attribute name.

Parameters

name	Name of attribute to find, or 0 to return previous attribute regardless of its name; this string doesn't have to be zero-terminated if name_size is non-zero
name_size	Size of name, in characters, or 0 to have size calculated automatically from string
case_sensitive	Should name comparison be case-sensitive; non case-sensitive comparison works properly only for ASCII characters

Returns

Pointer to found attribute, or 0 if not found.

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

- parsing/rapidxml-1.13/rapidxml.hpp

4.37 rapidxml::xml_base< Ch > Class Template Reference

`#include <rapidxml.hpp>`

Inherited by rapidxml::xml_attribute< Ch >, and rapidxml::xml_node< Ch >.

Public Member Functions

- Ch * name () const
- std::size_t name_size () const
- Ch * value () const
- std::size_t value_size () const
- void name (const Ch *name, std::size_t size)
- void name (const Ch *name)
- void value (const Ch *value, std::size_t size)
- void value (const Ch *value)
- xml_node< Ch > * parent () const

Static Protected Member Functions

- static Ch * **nullstr** ()

Protected Attributes

- Ch * **m_name**
- Ch * **m_value**
- std::size_t **m_name_size**
- std::size_t **m_value_size**
- xml_node< Ch > * **m_parent**

4.37.1 Detailed Description

template<class Ch = char>class rapidxml::xml_base< Ch >

Base class for xml_node and xml_attribute implementing common functions: name(), name_size(), value(), value_size() and parent().

Parameters

Ch	Character type to use

4.37.2 Member Function Documentation

4.37.2.1 template<class Ch = char> Ch* rapidxml::xml_base< Ch >::name ( ) const [inline]

Gets name of the node. Interpretation of name depends on type of node. Note that name will not be zero-terminated if rapidxml::parse_no_string_terminators option was selected during parse.

Use name_size() function to determine length of the name.

Returns

Name of node, or empty string if node has no name.

4.37.2.2 template<class Ch = char> void rapidxml::xml_base< Ch >::name (const Ch * *name*, std::size_t *size*) [inline]

Sets name of node to a non zero-terminated string. See ownership_of_strings.

Note that node does not own its name or value, it only stores a pointer to it. It will not delete or otherwise free the pointer on destruction. It is reponsibility of the user to properly manage lifetime of the string. The easiest way to achieve it is to use memory_pool of the document to allocate the string - on destruction of the document the string will be automatically freed.

Size of name must be specified separately, because name does not have to be zero terminated. Use name(const Ch *) function to have the length automatically calculated (string must be zero terminated).

Parameters

name	Name of node to set. Does not have to be zero terminated.
size	Size of name, in characters. This does not include zero terminator, if one is present.

4.37.2.3 template<class Ch = char> void rapidxml::xml_base< Ch >::name (const Ch * *name*) [inline]

Sets name of node to a zero-terminated string. See also ownership_of_strings and xml_node::name(const Ch *, std::size_t).

Parameters

name	Name of node to set. Must be zero terminated.

4.37.2.4 template<class Ch = char> std::size_t rapidxml::xml_base< Ch >::name_size ( ) const [inline]

Gets size of node name, not including terminator character. This function works correctly irrespective of whether name is or is not zero terminated.

Returns

Size of node name, in characters.

4.37.2.5 template<class Ch = char> xml_node<Ch>* rapidxml::xml_base< Ch >::parent ( ) const [inline]

Gets node parent.

Returns

Pointer to parent node, or 0 if there is no parent.

4.37.2.6 template<class Ch = char> Ch* rapidxml::xml_base< Ch >::value ( ) const [inline]

Gets value of node. Interpretation of value depends on type of node. Note that value will not be zero-terminated if rapidxml::parse_no_string_terminators option was selected during parse.

Use value_size() function to determine length of the value.

Returns

Value of node, or empty string if node has no value.

4.37.2.7 template<class Ch = char> void rapidxml::xml_base< Ch >::value (const Ch * *value*, std::size_t *size*) [inline]

Sets value of node to a non zero-terminated string. See ownership_of_strings.

Note that node does not own its name or value, it only stores a pointer to it. It will not delete or otherwise free the pointer on destruction. It is reponsibility of the user to properly manage lifetime of the string. The easiest way to achieve it is to use memory_pool of the document to allocate the string - on destruction of the document the string will be automatically freed.

Size of value must be specified separately, because it does not have to be zero terminated. Use value(const Ch *) function to have the length automatically calculated (string must be zero terminated).

If an element has a child node of type node_data, it will take precedence over element value when printing. If you want to manipulate data of elements using values, use parser flag rapidxml::parse_no_data_nodes to prevent creation of data nodes by the parser.

Parameters

value	value of node to set. Does not have to be zero terminated.
size	Size of value, in characters. This does not include zero terminator, if one is present.

4.37.2.8 template<class Ch = char> void rapidxml::xml_base< Ch >::value (const Ch * *value*) [inline]

Sets value of node to a zero-terminated string. See also ownership_of_strings and xml_node::value(const Ch *, std::size_t).

Parameters

value	Vame of node to set. Must be zero terminated.

4.37.2.9 template<class Ch = char> std::size_t rapidxml::xml_base< Ch >::value_size ( ) const [inline]

Gets size of node value, not including terminator character. This function works correctly irrespective of whether value is or is not zero terminated.

Returns

Size of node value, in characters.

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

- parsing/rapidxml-1.13/rapidxml.hpp

4.38 rapidxml::xml_document< Ch > Class Template Reference

`#include <rapidxml.hpp>`

Inherits rapidxml::xml_node< Ch >, and rapidxml::memory_pool< Ch >.

Classes

- struct **attribute_name_pred**
- struct **attribute_value_pred**
- struct **attribute_value_pure_pred**
- struct **node_name_pred**
- struct **text_pred**
- struct **text_pure_no_ws_pred**
- struct **text_pure_with_ws_pred**
- struct **whitespace_pred**

Public Member Functions

- xml_document ()

  Constructs empty XML document.
- template<int Flags>
  void parse (Ch *text)
- void clear ()

Additional Inherited Members

4.38.1 Detailed Description

template<class Ch = char>class rapidxml::xml_document< Ch >

This class represents root of the DOM hierarchy. It is also an xml_node and a memory_pool through public inheritance. Use parse() function to build a DOM tree from a zero-terminated XML text string. parse() function allocates memory for nodes and attributes by using functions of xml_document, which are inherited from memory_pool. To access root node of the document, use the document itself, as if it was an xml_node.

Parameters

Ch	Character type to use.

4.38.2 Member Function Documentation

4.38.2.1 template<class Ch = char> void rapidxml::xml_document< Ch >::clear ( ) [inline]

Clears the document by deleting all nodes and clearing the memory pool. All nodes owned by document pool are destroyed.

Reimplemented from rapidxml::memory_pool< Ch >.

4.38.2.2 template<class Ch = char> template<int Flags> void rapidxml::xml_document< Ch >::parse (Ch * *text*) [inline]

Parses zero-terminated XML string according to given flags. Passed string will be modified by the parser, unless rapidxml::parse_non_destructive flag is used. The string must persist for the lifetime of the document. In case of error, rapidxml::parse_error exception will be thrown.

If you want to parse contents of a file, you must first load the file into the memory, and pass pointer to its beginning. Make sure that data is zero-terminated.

Document can be parsed into multiple times. Each new call to parse removes previous nodes and attributes (if any), but does not clear memory pool.

Parameters

text	XML data to parse; pointer is non-const to denote fact that this data may be modified by the parser.

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

- parsing/rapidxml-1.13/rapidxml.hpp

4.39 rapidxml::xml_node< Ch > Class Template Reference

`#include <rapidxml.hpp>`

Inherits rapidxml::xml_base< Ch >.

Inherited by rapidxml::xml_document< Ch >.

Public Member Functions

- xml_node (node_type type)
- node_type type () const
- xml_document< Ch > * document () const
- xml_node< Ch > * first_node (const Ch *name=0, std::size_t name_size=0, bool case_sensitive=true) const
- xml_node< Ch > * last_node (const Ch *name=0, std::size_t name_size=0, bool case_sensitive=true) const

- xml_node< Ch > * previous_sibling (const Ch *name=0, std::size_t name_size=0, bool case_sensitive=true) const
- xml_node< Ch > * next_sibling (const Ch *name=0, std::size_t name_size=0, bool case_sensitive=true) const
- xml_attribute< Ch > * first_attribute (const Ch *name=0, std::size_t name_size=0, bool case_sensitive=true) const
- xml_attribute< Ch > * last_attribute (const Ch *name=0, std::size_t name_size=0, bool case_sensitive=true) const
- void type (node_type type)
- void prepend_node (xml_node< Ch > *child)
- void append_node (xml_node< Ch > *child)
- void insert_node (xml_node< Ch > *where, xml_node< Ch > *child)
- void remove_first_node ()
- void remove_last_node ()
- void remove_node (xml_node< Ch > *where)

  Removes specified child from the node.
- void remove_all_nodes ()

  Removes all child nodes (but not attributes).
- void prepend_attribute (xml_attribute< Ch > *attribute)
- void append_attribute (xml_attribute< Ch > *attribute)
- void insert_attribute (xml_attribute< Ch > *where, xml_attribute< Ch > *attribute)
- void remove_first_attribute ()
- void remove_last_attribute ()
- void remove_attribute (xml_attribute< Ch > *where)
- void remove_all_attributes ()

  Removes all attributes of node.

Additional Inherited Members

4.39.1 Detailed Description

template<class Ch = char>class rapidxml::xml_node< Ch >

Class representing a node of XML document. Each node may have associated name and value strings, which are available through name() and value() functions. Interpretation of name and value depends on type of the node. Type of node can be determined by using type() function.

Note that after parse, both name and value of node, if any, will point interior of source text used for parsing. Thus, this text must persist in the memory for the lifetime of node.

Parameters

Ch	Character type to use.

4.39.2 Constructor & Destructor Documentation

4.39.2.1 template<class Ch = char> rapidxml::xml_node< Ch >::xml_node (node_type *type*) [inline]

Constructs an empty node with the specified type. Consider using memory_pool of appropriate document to allocate nodes manually.

Parameters

type	Type of node to construct.

4.39.3 Member Function Documentation

4.39.3.1 template<class Ch = char> void rapidxml::xml_node< Ch >::append_attribute (xml_attribute< Ch > * *attribute*) [inline]

Appends a new attribute to the node.

Parameters

attribute	Attribute to append.

4.39.3.2 template<class Ch = char> void rapidxml::xml_node< Ch >::append_node (xml_node< Ch > * *child*) [inline]

Appends a new child node. The appended child becomes the last child.

Parameters

child	Node to append.

4.39.3.3 template<class Ch = char> xml_document<Ch>* rapidxml::xml_node< Ch >::document ( ) const [inline]

Gets document of which node is a child.

Returns

Pointer to document that contains this node, or 0 if there is no parent document.

4.39.3.4 template<class Ch = char> xml_attribute<Ch>* rapidxml::xml_node< Ch >::first_attribute (const Ch * *name* = 0, std::size_t *name_size* = 0, bool *case_sensitive* = true) const [inline]

Gets first attribute of node, optionally matching attribute name.

Parameters

name	Name of attribute to find, or 0 to return first attribute regardless of its name; this string doesn't have to be zero-terminated if name_size is non-zero
name_size	Size of name, in characters, or 0 to have size calculated automatically from string
case_sensitive	Should name comparison be case-sensitive; non case-sensitive comparison works properly only for ASCII characters

Returns

Pointer to found attribute, or 0 if not found.

4.39.3.5 template<class Ch = char> xml_node<Ch>* rapidxml::xml_node< Ch >::first_node (const Ch * *name* = 0, std::size_t *name_size* = 0, bool *case_sensitive* = true) const [inline]

Gets first child node, optionally matching node name.

Parameters

name	Name of child to find, or 0 to return first child regardless of its name; this string doesn't have to be zero-terminated if name_size is non-zero
name_size	Size of name, in characters, or 0 to have size calculated automatically from string
case_sensitive	Should name comparison be case-sensitive; non case-sensitive comparison works properly only for ASCII characters

Returns

Pointer to found child, or 0 if not found.

4.39.3.6 template<class Ch = char> void rapidxml::xml_node< Ch >::insert_attribute (xml_attribute< Ch > * *where*, xml_attribute< Ch > * *attribute*) [inline]

Inserts a new attribute at specified place inside the node. All attributes after and including the specified attribute are moved one position back.

Parameters

where	Place where to insert the attribute, or 0 to insert at the back.
attribute	Attribute to insert.

4.39.3.7 template<class Ch = char> void rapidxml::xml_node< Ch >::insert_node (xml_node< Ch > * *where*, xml_node< Ch > * *child*) [inline]

Inserts a new child node at specified place inside the node. All children after and including the specified node are moved one position back.

Parameters

where	Place where to insert the child, or 0 to insert at the back.
child	Node to insert.

4.39.3.8 template<class Ch = char> xml_attribute<Ch>* rapidxml::xml_node< Ch >::last_attribute (const Ch * *name* = 0, std::size_t *name_size* = 0, bool *case_sensitive* = true) const [inline]

Gets last attribute of node, optionally matching attribute name.

Parameters

name	Name of attribute to find, or 0 to return last attribute regardless of its name; this string doesn't have to be zero-terminated if name_size is non-zero
name_size	Size of name, in characters, or 0 to have size calculated automatically from string
case_sensitive	Should name comparison be case-sensitive; non case-sensitive comparison works properly only for ASCII characters

Returns

Pointer to found attribute, or 0 if not found.

4.39.3.9 template<class Ch = char> xml_node<Ch>* rapidxml::xml_node< Ch >::last_node (const Ch * *name* = 0, std::size_t *name_size* = 0, bool *case_sensitive* = true) const [inline]

Gets last child node, optionally matching node name. Behaviour is undefined if node has no children. Use first_node() to test if node has children.

Parameters

name	Name of child to find, or 0 to return last child regardless of its name; this string doesn't have to be zero-terminated if name_size is non-zero
name_size	Size of name, in characters, or 0 to have size calculated automatically from string
case_sensitive	Should name comparison be case-sensitive; non case-sensitive comparison works properly only for ASCII characters

Returns

Pointer to found child, or 0 if not found.

4.39.3.10 template<class Ch = char> xml_node<Ch>* rapidxml::xml_node< Ch >::next_sibling (const Ch * *name* = 0, std::size_t *name_size* = 0, bool *case_sensitive* = true) const [inline]

Gets next sibling node, optionally matching node name. Behaviour is undefined if node has no parent. Use parent() to test if node has a parent.

Parameters

name	Name of sibling to find, or 0 to return next sibling regardless of its name; this string doesn't have to be zero-terminated if name_size is non-zero
name_size	Size of name, in characters, or 0 to have size calculated automatically from string
case_sensitive	Should name comparison be case-sensitive; non case-sensitive comparison works properly only for ASCII characters

Returns

Pointer to found sibling, or 0 if not found.

4.39.3.11 template<class Ch = char> void rapidxml::xml_node< Ch >::prepend_attribute (xml_attribute< Ch > * *attribute*) [inline]

Prepends a new attribute to the node.

Parameters

attribute	Attribute to prepend.

4.39.3.12 template<class Ch = char> void rapidxml::xml_node< Ch >::prepend_node (xml_node< Ch > * *child*) [inline]

Prepends a new child node. The prepended child becomes the first child, and all existing children are moved one position back.

Parameters

child	Node to prepend.

4.39.3.13 template<class Ch = char> xml_node<Ch>* rapidxml::xml_node< Ch >::previous_sibling (const Ch * *name* = 0, std::size_t *name_size* = 0, bool *case_sensitive* = true) const [inline]

Gets previous sibling node, optionally matching node name. Behaviour is undefined if node has no parent. Use parent() to test if node has a parent.

Parameters

name	Name of sibling to find, or 0 to return previous sibling regardless of its name; this string doesn't have to be zero-terminated if name_size is non-zero
name_size	Size of name, in characters, or 0 to have size calculated automatically from string
case_sensitive	Should name comparison be case-sensitive; non case-sensitive comparison works properly only for ASCII characters

Returns

Pointer to found sibling, or 0 if not found.

4.39.3.14 template<class Ch = char> void rapidxml::xml_node< Ch >::remove_attribute (xml_attribute< Ch > * *where*) [inline]

Removes specified attribute from node.

Parameters

where	Pointer to attribute to be removed.

4.39.3.15 template<class Ch = char> void rapidxml::xml_node< Ch >::remove_first_attribute ( ) [inline]

Removes first attribute of the node. If node has no attributes, behaviour is undefined. Use first_attribute() to test if node has attributes.

4.39.3.16 template<class Ch = char> void rapidxml::xml_node< Ch >::remove_first_node ( ) [inline]

Removes first child node. If node has no children, behaviour is undefined. Use first_node() to test if node has children.

4.39.3.17 template<class Ch = char> void rapidxml::xml_node< Ch >::remove_last_attribute ( ) [inline]

Removes last attribute of the node. If node has no attributes, behaviour is undefined. Use first_attribute() to test if node has attributes.

4.39.3.18 template<class Ch = char> void rapidxml::xml_node< Ch >::remove_last_node ( ) [inline]

Removes last child of the node. If node has no children, behaviour is undefined. Use first_node() to test if node has children.

4.39.3.19 template<class Ch = char> node_type rapidxml::xml_node< Ch >::type ( ) const [inline]

Gets type of node.

Returns

Type of node.

4.39.3.20 template<class Ch = char> void rapidxml::xml_node< Ch >::type (node_type *type*) [inline]

Sets type of node.

Parameters

type	Type of node to set.

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

- parsing/rapidxml-1.13/rapidxml.hpp

Chapter 5

File Documentation

5.1 parsing/rapidxml-1.13/rapidxml.hpp File Reference

This file contains rapidxml parser and DOM implementation.

```
#include <cstdlib>
#include <cassert>
#include <new>
#include <iterator>
#include <exception>
```

Classes

- class rapidxml::parse_error
- class rapidxml::memory_pool< Ch >
- struct **rapidxml::memory_pool< Ch >::header**
- class rapidxml::xml_base< Ch >
- class rapidxml::xml_attribute< Ch >
- class rapidxml::xml_node< Ch >
- class rapidxml::xml_document< Ch >
- struct **rapidxml::xml_document< Ch >::whitespace_pred**
- struct **rapidxml::xml_document< Ch >::node_name_pred**
- struct **rapidxml::xml_document< Ch >::attribute_name_pred**
- struct **rapidxml::xml_document< Ch >::text_pred**
- struct **rapidxml::xml_document< Ch >::text_pure_no_ws_pred**
- struct **rapidxml::xml_document< Ch >::text_pure_with_ws_pred**
- struct **rapidxml::xml_document< Ch >::attribute_value_pred< Quote >**
- struct **rapidxml::xml_document< Ch >::attribute_value_pure_pred< Quote >**

Macros

- #define **RAPIDXML_PARSE_ERROR**(what, where) throw parse_error(what, where)
- #define **RAPIDXML_STATIC_POOL_SIZE** (64 * 1024)
- #define **RAPIDXML_DYNAMIC_POOL_SIZE** (64 * 1024)
- #define **RAPIDXML_ALIGNMENT** sizeof(void *)

Enumerations

- enum **node_type** {
  rapidxml::node_document, **rapidxml::node_element**, **rapidxml::node_data**, **rapidxml::node_cdata**,
  rapidxml::node_comment, **rapidxml::node_declaration**, **rapidxml::node_doctype**, **rapidxml::node_pi** }

Variables

- const int **rapidxml::parse_no_data_nodes** = 0x1
- const int **rapidxml::parse_no_element_values** = 0x2
- const int **rapidxml::parse_no_string_terminators** = 0x4
- const int **rapidxml::parse_no_entity_translation** = 0x8
- const int **rapidxml::parse_no_utf8** = 0x10
- const int **rapidxml::parse_declaration_node** = 0x20
- const int **rapidxml::parse_comment_nodes** = 0x40
- const int **rapidxml::parse_doctype_node** = 0x80
- const int **rapidxml::parse_pi_nodes** = 0x100
- const int **rapidxml::parse_validate_closing_tags** = 0x200
- const int **rapidxml::parse_trim_whitespace** = 0x400
- const int **rapidxml::parse_normalize_whitespace** = 0x800
- const int **rapidxml::parse_default** = 0
- const int **rapidxml::parse_non_destructive** = parse_no_string_terminators | parse_no_entity_translation
- const int **rapidxml::parse_fastest** = parse_non_destructive | parse_no_data_nodes
- const int **rapidxml::parse_full** = parse_declaration_node | parse_comment_nodes | parse_doctype_node | parse_pi_nodes | parse_validate_closing_tags

5.1.1 Detailed Description

This file contains rapidxml parser and DOM implementation.

5.2 parsing/rapidxml-1.13/rapidxml_iterators.hpp File Reference

This file contains rapidxml iterators.

```
#include "rapidxml.hpp"
```

Classes

- class rapidxml::node_iterator< Ch >

  Iterator of child nodes of xml_node.
- class rapidxml::attribute_iterator< Ch >

  Iterator of child attributes of xml_node.

5.2.1 Detailed Description

This file contains rapidxml iterators.

5.3 parsing/rapidxml-1.13/rapidxml_print.hpp File Reference

This file contains rapidxml printer implementation.

```
#include "rapidxml.hpp"
#include <ostream>
#include <iterator>
```

Functions

- template<class OutIt , class Ch >
  OutIt **rapidxml::print** (OutIt out, const xml_node< Ch > &node, int flags=0)
- template<class Ch >
  std::basic_ostream< Ch > & **rapidxml::print** (std::basic_ostream< Ch > &out, const xml_node< Ch > &node, int flags=0)
- template<class Ch >
  std::basic_ostream< Ch > & **rapidxml::operator**<< (std::basic_ostream< Ch > &out, const xml_node< Ch > &node)

Variables

- const int **rapidxml::print_no_indenting** = 0x1

  Printer flag instructing the printer to suppress indenting of XML. See print() function.

5.3.1 Detailed Description

This file contains rapidxml printer implementation.

5.4 parsing/rapidxml-1.13/rapidxml_utils.hpp File Reference

```
#include "rapidxml.hpp"
#include <vector>
#include <string>
#include <fstream>
#include <stdexcept>
```

Classes

- class rapidxml::file< Ch >

  Represents data loaded from a file.

Functions

- template<class Ch >
  std::size_t **rapidxml::count_children** (xml_node< Ch > *node)
- template<class Ch >
  std::size_t **rapidxml::count_attributes** (xml_node< Ch > *node)

5.4.1 Detailed Description

This file contains high-level rapidxml utilities that can be useful in certain simple scenarios. They should probably not be used if maximizing performance is the main objective.

Index