

# spot Reference Manual

## 0.1

Generated by Doxygen 1.4.0

Mon Jan 31 12:54:24 2005

## Contents

<b>1</b>	<b><a href="#">spot Main Page</a></b>	<b>1</b>
<b>2</b>	<b><a href="#">spot Module Index</a></b>	<b>2</b>
<b>3</b>	<b><a href="#">spot Directory Hierarchy</a></b>	<b>3</b>
<b>4</b>	<b><a href="#">spot Namespace Index</a></b>	<b>3</b>
<b>5</b>	<b><a href="#">spot Hierarchical Index</a></b>	<b>4</b>
<b>6</b>	<b><a href="#">spot Class Index</a></b>	<b>8</b>
<b>7</b>	<b><a href="#">spot File Index</a></b>	<b>13</b>
<b>8</b>	<b><a href="#">spot Module Documentation</a></b>	<b>17</b>
<b>9</b>	<b><a href="#">spot Directory Documentation</a></b>	<b>50</b>
<b>10</b>	<b><a href="#">spot Namespace Documentation</a></b>	<b>57</b>
<b>11</b>	<b><a href="#">spot Class Documentation</a></b>	<b>76</b>
<b>12</b>	<b><a href="#">spot File Documentation</a></b>	<b>412</b>

## 1 spot Main Page

### 1.1 The Spot Library

Spot is a model-checking library. It provides algorithms and data structures to implement the automata-theoretic approach to model-checking.

See [spot.lip6.fr](http://spot.lip6.fr) for more information about this project.

### 1.2 This Document

This document describes all the public data structures and functions of Spot. This aims to be a reference manual, not a tutorial.

If you are new to this manual, start with [the module page](#). This is what looks the closest to a table of contents.

### 1.3 Handy starting points

- [spot::ltl::formula](#) Base class for an LTL formulae.
- [spot::ltl::parse](#) Parsing a text string into a [spot::ltl::formula](#).

- [spot::tgba](#) Base class for Transition-based Generalized Büchi Automaton.
- [spot::ltl\\_to\\_tgba\\_fm](#) Convert a [spot::ltl::formula](#) into a [spot::tgba](#).
- [spot::tgba\\_product](#) On-the-fly product of two [spot::tgba](#).
- [spot::emptiness\\_check](#) Base class for all emptiness-check algorithms (see also module [Emptiness-checks](#))

## 2 spot Module Index

### 2.1 spot Modules

Here is a list of all modules:

<b>LTL formulae</b>	<b>17</b>
<b>Essential LTL types</b>	<b>18</b>
<b>LTL Abstract Syntax Tree</b>	<b>18</b>
<b>LTL environments</b>	<b>19</b>
<b>Algorithms for LTL formulae</b>	<b>19</b>
<b>Input/Output of LTL formulae</b>	<b>19</b>
<b>Derivable visitors</b>	<b>22</b>
<b>Rewriting LTL formulae</b>	<b>22</b>
<b>Miscellaneous algorithms for LTL formulae</b>	<b>24</b>
<b>TGBA (Transition-based Generalized Büchi Automata)</b>	<b>17</b>
<b>Essential TGBA types</b>	<b>31</b>
<b>TGBA representations</b>	<b>31</b>
<b>TGBA algorithms</b>	<b>32</b>
<b>TGBA on-the-fly algorithms</b>	<b>32</b>
<b>Input/Output of TGBA</b>	<b>32</b>
<b>Decorating the dot output</b>	<b>41</b>
<b>Translating LTL formulae into TGBA</b>	<b>35</b>
<b>Algorithm patterns</b>	<b>36</b>
<b>TGBA simplifications</b>	<b>37</b>
<b>Miscellaneous algorithms on TGBA</b>	<b>39</b>
<b>Emptiness-checks</b>	<b>41</b>

Emptiness-check algorithms for SSP	17
Emptiness-check algorithms	42
TGBA runs and supporting functions	48
Emptiness-check statistics	50
Miscellaneous helper algorithms	27
Hashing functions	28
Random functions	29

## 3 spot Directory Hierarchy

### 3.1 spot Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

evtgba	50
evtgbalogs	51
evtgbaparse	51
gspn	51
ltlast	52
ltlenv	53
ltlparse	54
ltlvisit	54
misc	55
tgba	55
tgbaalogs	56
gtec	52
tgbaparse	57

## 4 spot Namespace Index

### 4.1 spot Namespace List

Here is a list of all namespaces with brief descriptions:

spot	57
------	----

<a href="#">spot::ltl</a>	70
<a href="#">yy</a>	74

## 5 spot Hierarchical Index

### 5.1 spot Class Hierarchy

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

<a href="#">spot::acss_statistics</a>	76
<a href="#">spot::couvreur99_check_result</a>	129
<a href="#">spot::ars_statistics</a>	76
<a href="#">spot::couvreur99_check_result</a>	129
<a href="#">spot::barand&lt; gen &gt;</a>	84
<a href="#">spot::bdd_less_than</a>	102
<a href="#">spot::bfs_steps</a>	103
<a href="#">spot::ltl::const_visitor</a>	119
<a href="#">spot::couvreur99_check_shy::successor</a>	138
<a href="#">spot::couvreur99_check_shy::todo_item</a>	139
<a href="#">spot::couvreur99_check_status</a>	140
<a href="#">spot::delayed_simulation_relation</a>	146
<a href="#">spot::direct_simulation_relation</a>	146
<a href="#">spot::dotty_decorator</a>	146
<a href="#">spot::tgba_run_dotty_decorator</a>	364
<a href="#">spot::ec_statistics</a>	157
<a href="#">spot::couvreur99_check</a>	124
<a href="#">spot::couvreur99_check_shy</a>	132
<a href="#">spot::emptiness_check</a>	160
<a href="#">spot::couvreur99_check</a>	124
<a href="#">spot::emptiness_check_result</a>	162
<a href="#">spot::couvreur99_check_result</a>	129
<a href="#">spot::ltl::environment</a>	164

spot::ltl::declarative_environment	141
spot::ltl::default_environment	144
spot::evtgba	166
spot::evtgba_explicit	168
spot::evtgba_product	175
spot::evtgba_explicit::state	172
spot::evtgba_explicit::transition	173
spot::evtgba_iterator	174
spot::evtgba_reachable_iterator	178
spot::evtgba_reachable_iterator_breadth_first	181
spot::evtgba_reachable_iterator_depth_first	185
spot::explicit_connected_component_factory	191
spot::connected_component_hash_set_factory	117
spot::ltl::formula	191
spot::ltl::constant	120
spot::ltl::ref_formula	255
spot::ltl::atomic_prop	78
spot::ltl::binop	105
spot::ltl::multop	212
spot::ltl::unop	402
spot::ltl::formula_ptr_hash	195
spot::ltl::formula_ptr_less_than	195
spot::free_list	196
spot::bdd_allocator	85
spot::bdd_dict	90
spot::bdd_dict::annon_free_list	99
spot::gspn_exeption	199
spot::gspn_interface	200
spot::gspn_ssp_interface	202

<code>yy::Location</code>	204
<code>spot::loopless_modular_mixed_radix_gray_code</code>	206
<code>spot::minato_isop</code>	209
<code>spot::minato_isop::local_vars</code>	211
<code>spot::lftl::multop::paircmp</code>	219
<code>spot::numbered_state_heap</code>	220
<code>spot::numbered_state_heap_hash_map</code>	225
<code>spot::numbered_state_heap_const_iterator</code>	223
<code>spot::numbered_state_heap_factory</code>	224
<code>spot::numbered_state_heap_hash_map_factory</code>	228
<code>yy::Position</code>	247
<code>spot::ptr_hash&lt; T &gt;</code>	251
<code>spot::lftl::random_lftl</code>	251
<code>spot::lftl::random_lftl::op_proba</code>	254
<code>spot::rsymbol</code>	259
<code>spot::scc_stack</code>	261
<code>spot::scc_stack::connected_component</code>	263
<code>spot::explicit_connected_component</code>	189
<code>spot::connected_component_hash_set</code>	115
<code>yy::Slice&lt; T, S &gt;</code>	267
<code>spot::spoiler_node</code>	268
<code>spot::duplicator_node</code>	148
<code>spot::duplicator_node_delayed</code>	153
<code>spot::spoiler_node_delayed</code>	270
<code>yy::Stack&lt; T, S &gt;</code>	274
<code>spot::state</code>	276
<code>spot::state_bdd</code>	278
<code>spot::state_evtgba_explicit</code>	280
<code>spot::state_explicit</code>	283

spot::state_product	286
spot::state_ptr_equal	289
spot::state_ptr_hash	289
spot::state_ptr_less_than	290
spot::string_hash	290
spot::symbol	291
spot::tgba	293
spot::tgba_bdd_concrete	299
spot::tgba_explicit	315
spot::tgba_reduc	347
spot::tgba_product	328
spot::tgba_tba_proxy	384
spot::tgba_sba_proxy	367
spot::tgba_bdd_core_data	309
spot::tgba_bdd_factory	314
spot::tgba_bdd_concrete_factory	305
spot::tgba_explicit::transition	324
spot::tgba_reachable_iterator	335
spot::tgba_reachable_iterator_breadth_first	339
spot::parity_game_graph	230
spot::parity_game_graph_delayed	235
spot::parity_game_graph_direct	241
spot::tgba_reduc	347
spot::tgba_reachable_iterator_depth_first	343
spot::tgba_run	362
spot::tgba_run::step	363
spot::tgba_statistics	373
spot::tgba_succ_iterator	373
spot::tgba_explicit_succ_iterator	325



spot::tgba_succ_iterator_concrete	376
spot::tgba_succ_iterator_product	380
spot::time_info	391
spot::timer	392
spot::timer_map	393
spot::ltl::visitor	409
spot::ltl::clone_visitor	112
spot::ltl::simplify_f_g_visitor	264
spot::ltl::unabbreviate_logic_visitor	395
spot::ltl::unabbreviate_ltl_visitor	399
spot::ltl::postfix_visitor	249
spot::weight	410

## 6 spot Class Index

### 6.1 spot Class List

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

spot::acss_statistics (Accepting Cycle Search Space statistics )	76
spot::ars_statistics (Accepting Run Search statistics )	76
spot::ltl::atomic_prop (Atomic propositions )	78
spot::barand< gen > (Compute pseudo-random integer value between 0 and $n$ included, following a binomial distribution for probability $p$ )	84
spot::bdd_allocator (Manage ranges of variables )	85
spot::bdd_dict	90
spot::bdd_dict::annon_free_list	99
spot::bdd_less_than (Comparison functor for BDDs )	102
spot::bfs_steps (Make a BFS in a <code>spot::tgba</code> to compute a <code>tgba_run::steps</code> )	103
spot::ltl::binop (Binary operator )	105
spot::ltl::clone_visitor (Clone a formula )	112
spot::connected_component_hash_set	115
spot::connected_component_hash_set_factory (Factory for <code>connected_component_hash_set</code> )	117

<a href="#">spot::ltl::const_visitor</a> (Formula visitor that cannot modify the formula )	119
<a href="#">spot::ltl::constant</a> (A constant (True or False) )	120
<a href="#">spot::couvereur99_check</a> (Check whether the language of an automate is empty )	124
<a href="#">spot::couvereur99_check_result</a> (Compute a counter example from a <a href="#">spot::couvereur99_check_status</a> )	129
<a href="#">spot::couvereur99_check_shy</a> (A version of <a href="#">spot::couvereur99_check</a> that tries to visit known states first )	132
<a href="#">spot::couvereur99_check_shy::successor</a>	138
<a href="#">spot::couvereur99_check_shy::todo_item</a>	139
<a href="#">spot::couvereur99_check_status</a> (The status of the emptiness-check on success )	140
<a href="#">spot::ltl::declarative_environment</a> (A declarative environment )	141
<a href="#">spot::ltl::default_environment</a> (A laxist environment )	144
<a href="#">spot::delayed_simulation_relation</a>	146
<a href="#">spot::direct_simulation_relation</a>	146
<a href="#">spot::dotty_decorator</a> (Choose state and link styles for <a href="#">spot::dotty_reachable</a> )	146
<a href="#">spot::duplicator_node</a> (Duplicator node of parity game graph )	148
<a href="#">spot::duplicator_node_delayed</a> (Duplicator node of parity game graph for delayed simulation )	153
<a href="#">spot::ec_statistics</a> (Emptiness-check statistics )	157
<a href="#">spot::emptiness_check</a> (Common interface to emptiness check algorithms )	160
<a href="#">spot::emptiness_check_result</a> (The result of an emptiness check )	162
<a href="#">spot::ltl::environment</a> (An environment that describes atomic propositions )	164
<a href="#">spot::evtgba</a>	166
<a href="#">spot::evtgba_explicit</a>	168
<a href="#">spot::evtgba_explicit::state</a>	172
<a href="#">spot::evtgba_explicit::transition</a> (Explicit transitions (used by <a href="#">spot::evtgba_explicit</a> ) )	173
<a href="#">spot::evtgba_iterator</a>	174
<a href="#">spot::evtgba_product</a>	175
<a href="#">spot::evtgba_reachable_iterator</a> (Iterate over all reachable states of a <a href="#">spot::evtgba</a> )	178
<a href="#">spot::evtgba_reachable_iterator_breadth_first</a> (An implementation of <a href="#">spot::evtgba_reachable_iterator</a> that browses states breadth first )	181

<a href="#">spot::evtgba_reachable_iterator_depth_first</a> (An implementation of <a href="#">spot::evtgba_reachable_iterator</a> that browses states depth first )	185
<a href="#">spot::explicit_connected_component</a> (An SCC storing all its states explicitly )	189
<a href="#">spot::explicit_connected_component_factory</a> (Abstract factory for <a href="#">explicit_connected_component</a> )	191
<a href="#">spot::ltl::formula</a> (An LTL formula )	191
<a href="#">spot::ltl::formula_ptr_hash</a> (Hash Function for <code>const formula*</code> )	195
<a href="#">spot::ltl::formula_ptr_less_than</a> (Strict Weak Ordering for <code>const formula*</code> )	195
<a href="#">spot::free_list</a> (Manage list of free integers )	196
<a href="#">spot::gspn_exemption</a> (An exemption used to forward GSPN errors )	199
<a href="#">spot::gspn_interface</a>	200
<a href="#">spot::gspn_ssp_interface</a>	202
<a href="#">yy::Location</a> (Abstract a <b>Location</b> )	204
<a href="#">spot::loopless_modular_mixed_radix_gray_code</a> (Loopless modular mixed radix Gray code iteration )	206
<a href="#">spot::minato_isop</a> (Generate an irredundant sum-of-products (ISOP) form of a BDD function )	209
<a href="#">spot::minato_isop::local_vars</a> (Internal variables for <a href="#">minato_isop</a> )	211
<a href="#">spot::ltl::multop</a> (Multi-operand operators )	212
<a href="#">spot::ltl::multop::pairemp</a> (Comparison functor used internally by <a href="#">ltl::multop</a> )	219
<a href="#">spot::numbered_state_heap</a> (Keep track of a large quantity of indexed states )	220
<a href="#">spot::numbered_state_heap_const_iterator</a> (Iterator on <a href="#">numbered_state_heap</a> objects )	223
<a href="#">spot::numbered_state_heap_factory</a> (Abstract factory for <a href="#">numbered_state_heap</a> )	224
<a href="#">spot::numbered_state_heap_hash_map</a> (A straightforward implementation of <a href="#">numbered_state_heap</a> with a hash map )	225
<a href="#">spot::numbered_state_heap_hash_map_factory</a> (Factory for <a href="#">numbered_state_heap_hash_map</a> )	228
<a href="#">spot::parity_game_graph</a> (Parity game graph which compute a simulation relation )	230
<a href="#">spot::parity_game_graph_delayed</a>	235
<a href="#">spot::parity_game_graph_direct</a> (Parity game graph which compute the direct simulation relation )	241
<a href="#">yy::Position</a> (Abstract a <b>Position</b> )	247

<a href="#">spot::ltl::postfix_visitor</a> (Apply an algorithm on each node of an AST, during a postfix traversal )	249
<a href="#">spot::ptr_hash&lt; T &gt;</a> (A hash function for pointers )	251
<a href="#">spot::ltl::random_ltl</a> (Generate random LTL formulae )	251
<a href="#">spot::ltl::random_ltl::op_proba</a>	254
<a href="#">spot::ltl::ref_formula</a> (A reference-counted LTL formula )	255
<a href="#">spot::rsymbol</a>	259
<a href="#">spot::scc_stack</a>	261
<a href="#">spot::scc_stack::connected_component</a>	263
<a href="#">spot::ltl::simplify_f_g_visitor</a> (Replace <code>true U f</code> and <code>false R g</code> by <code>F f</code> and <code>G g</code> )	264
<a href="#">yy::Slice&lt; T, S &gt;</a>	267
<a href="#">spot::spoiler_node</a> (Spoiler node of parity game graph )	268
<a href="#">spot::spoiler_node_delayed</a> (Spoiler node of parity game graph for delayed simulation )	270
<a href="#">yy::Stack&lt; T, S &gt;</a>	274
<a href="#">spot::state</a> (Abstract class for states )	276
<a href="#">spot::state_bdd</a>	278
<a href="#">spot::state_evtgba_explicit</a> (States used by <a href="#">spot::tgba_evtgba_explicit</a> )	280
<a href="#">spot::state_explicit</a>	283
<a href="#">spot::state_product</a> (A state for <a href="#">spot::tgba_product</a> )	286
<a href="#">spot::state_ptr_equal</a> (An Equivalence Relation for <code>state*</code> )	289
<a href="#">spot::state_ptr_hash</a> (Hash Function for <code>state*</code> )	289
<a href="#">spot::state_ptr_less_than</a> (Strict Weak Ordering for <code>state*</code> )	290
<a href="#">spot::string_hash</a> (A hash function for strings )	290
<a href="#">spot::symbol</a>	291
<a href="#">spot::tgba</a> (A Transition-based Generalized Büchi Automaton )	293
<a href="#">spot::tgba_bdd_concrete</a> (A concrete <a href="#">spot::tgba</a> implemented using BDDs )	299
<a href="#">spot::tgba_bdd_concrete_factory</a> (Helper class to build a <a href="#">spot::tgba_bdd_concrete</a> object )	305
<a href="#">spot::tgba_bdd_core_data</a> (Core data for a TGBA encoded using BDDs )	309
<a href="#">spot::tgba_bdd_factory</a> (Abstract class for <a href="#">spot::tgba_bdd_concrete</a> factories )	314
<a href="#">spot::tgba_explicit</a>	315

<a href="#">spot::tgba_explicit::transition</a> (Explicit transitions (used by <a href="#">spot::tgba_explicit</a> ) )	324
<a href="#">spot::tgba_explicit_succ_iterator</a>	325
<a href="#">spot::tgba_product</a> (A lazy product. (States are computed on the fly.) )	328
<a href="#">spot::tgba_reachable_iterator</a> (Iterate over all reachable states of a <a href="#">spot::tgba</a> )	335
<a href="#">spot::tgba_reachable_iterator_breadth_first</a> (An implementation of <a href="#">spot::tgba_reachable_iterator</a> that browses states breadth first )	339
<a href="#">spot::tgba_reachable_iterator_depth_first</a> (An implementation of <a href="#">spot::tgba_reachable_iterator</a> that browses states depth first )	343
<a href="#">spot::tgba_reduc</a>	347
<a href="#">spot::tgba_run</a> (An accepted run, for a <a href="#">tgba</a> )	362
<a href="#">spot::tgba_run::step</a>	363
<a href="#">spot::tgba_run_dotty_decorator</a> (Highlight a <a href="#">spot::tgba_run</a> on a <a href="#">spot::tgba</a> )	364
<a href="#">spot::tgba_sba_proxy</a> (Degeneralize a <a href="#">spot::tgba</a> on the fly, producing an SBA )	367
<a href="#">spot::tgba_statistics</a>	373
<a href="#">spot::tgba_succ_iterator</a> (Iterate over the successors of a state )	373
<a href="#">spot::tgba_succ_iterator_concrete</a>	376
<a href="#">spot::tgba_succ_iterator_product</a> (Iterate over the successors of a product computed on the fly )	380
<a href="#">spot::tgba_tba_proxy</a> (Degeneralize a <a href="#">spot::tgba</a> on the fly, producing a TBA )	384
<a href="#">spot::time_info</a> (A structure to record elapsed time in clock ticks )	391
<a href="#">spot::timer</a> (A timekeeper that accumulate interval of time )	392
<a href="#">spot::timer_map</a> (A map of timer, where each timer has a name )	393
<a href="#">spot::ltl::unabbreviate_logic_visitor</a> (Clone and rewrite a formula to remove most of the abbreviated logical operators )	395
<a href="#">spot::ltl::unabbreviate_ltl_visitor</a> (Clone and rewrite a formula to remove most of the abbreviated LTL and logical operators )	399
<a href="#">spot::ltl::unop</a> (Unary operators )	402
<a href="#">spot::ltl::visitor</a> (Formula visitor that can modify the formula )	409
<a href="#">spot::weight</a> (Manage for a given automaton a vector of counter indexed by its acceptance condition )	410

## 7 spot File Index

### 7.1 spot File List

Here is a list of all files with brief descriptions:

<a href="#">gspn/common.hh</a>	412
<a href="#">gspn/gspn.hh</a>	413
<a href="#">gspn/ssp.hh</a>	414
<a href="#">evtgba/evtgba.hh</a>	414
<a href="#">evtgba/evtgbaiter.hh</a>	415
<a href="#">evtgba/explicit.hh</a>	416
<a href="#">evtgba/product.hh</a>	417
<a href="#">evtgba/symbol.hh</a>	418
<a href="#">evtgbaalgorithms/dotty.hh</a>	418
<a href="#">evtgbaalgorithms/reachiter.hh</a>	420
<a href="#">evtgbaalgorithms/save.hh</a>	422
<a href="#">evtgbaalgorithms/tgba2evtgba.hh</a>	423
<a href="#">evtgbaparse/public.hh</a>	424
<a href="#">ltlast/allnodes.hh</a> (Define all LTL node types )	429
<a href="#">ltlast/atomic_prop.hh</a> (LTL atomic propositions )	429
<a href="#">ltlast/binop.hh</a> (LTL binary operators )	430
<a href="#">ltlast/constant.hh</a> (LTL constants )	431
<a href="#">ltlast/formula.hh</a> (LTL formula interface )	431
<a href="#">ltlast/multop.hh</a> (LTL multi-operand operators )	433
<a href="#">ltlast/predecl.hh</a> (Predeclare all LTL node types )	433
<a href="#">ltlast/refformula.hh</a> (Reference-counted LTL formulae )	434
<a href="#">ltlast/unop.hh</a> (LTL unary operators )	434
<a href="#">ltlast/visitor.hh</a> (LTL visitor interface )	435
<a href="#">ltlenv/declenv.hh</a>	436
<a href="#">ltlenv/defaultenv.hh</a>	436
<a href="#">ltlenv/environment.hh</a>	437

<a href="#">ltlparse/location.hh</a>	437
<a href="#">ltlparse/position.hh</a>	439
<a href="#">ltlparse/public.hh</a>	426
<a href="#">ltlparse/stack.hh</a>	440
<a href="#">ltlvisit/apcollect.hh</a>	441
<a href="#">ltlvisit/basicreduce.hh</a>	441
<a href="#">ltlvisit/clone.hh</a>	442
<a href="#">ltlvisit/destroy.hh</a>	443
<a href="#">ltlvisit/dotty.hh</a>	419
<a href="#">ltlvisit/dump.hh</a>	443
<a href="#">ltlvisit/length.hh</a>	444
<a href="#">ltlvisit/lunabbrev.hh</a>	444
<a href="#">ltlvisit/nenoform.hh</a>	445
<a href="#">ltlvisit/postfix.hh</a>	445
<a href="#">ltlvisit/randomltl.hh</a>	446
<a href="#">ltlvisit/reduce.hh</a>	446
<a href="#">ltlvisit/simpfg.hh</a>	447
<a href="#">ltlvisit/syntimpl.hh</a>	447
<a href="#">ltlvisit/tostring.hh</a>	448
<a href="#">ltlvisit/tunabbrev.hh</a>	449
<a href="#">misc/bareword.hh</a>	449
<a href="#">misc/bddalloc.hh</a>	450
<a href="#">misc/bddltt.hh</a>	450
<a href="#">misc/escape.hh</a>	451
<a href="#">misc/freelist.hh</a>	451
<a href="#">misc/hash.hh</a>	452
<a href="#">misc/hashfunc.hh</a>	452
<a href="#">misc/minato.hh</a>	453
<a href="#">misc/modgray.hh</a>	453

<a href="#">misc/random.hh</a>	453
<a href="#">misc/timer.hh</a>	454
<a href="#">misc/version.hh</a>	455
<a href="#">tgba/bdddict.hh</a>	455
<a href="#">tgba/bddprint.hh</a>	456
<a href="#">tgba/formula2bdd.hh</a>	460
<a href="#">tgba/public.hh</a>	427
<a href="#">tgba/state.hh</a>	461
<a href="#">tgba/statebdd.hh</a>	462
<a href="#">tgba/succiter.hh</a>	462
<a href="#">tgba/succiterconcrete.hh</a>	463
<a href="#">tgba/tgba.hh</a>	464
<a href="#">tgba/tgababddconcrete.hh</a>	466
<a href="#">tgba/tgababddconcretefactory.hh</a>	466
<a href="#">tgba/tgababddconcreteproduct.hh</a>	467
<a href="#">tgba/tgababddcoredata.hh</a>	468
<a href="#">tgba/tgababddfacyory.hh</a>	468
<a href="#">tgba/tgbaexplicit.hh</a>	469
<a href="#">tgba/tgbaproduct.hh</a>	470
<a href="#">tgba/tgbareduc.hh</a>	471
<a href="#">tgba/tgbatba.hh</a>	472
<a href="#">tgbaalgos/bfssteps.hh</a>	473
<a href="#">tgbaalgos/dotty.hh</a>	420
<a href="#">tgbaalgos/dottydec.hh</a>	474
<a href="#">tgbaalgos/dupep.hh</a>	474
<a href="#">tgbaalgos/emptiness.hh</a>	475
<a href="#">tgbaalgos/emptiness_stats.hh</a>	476
<a href="#">tgbaalgos/gv04.hh</a>	482
<a href="#">tgbaalgos/lbtt.hh</a>	482



tgbaalgos/ltl2tgba_fm.hh	482
tgbaalgos/ltl2tgba_lacim.hh	483
tgbaalgos/magic.hh	484
tgbaalgos/neverclaim.hh	484
tgbaalgos/powerset.hh	485
tgbaalgos/projrun.hh	486
tgbaalgos/randomgraph.hh	487
tgbaalgos/reachiter.hh	421
tgbaalgos/reducerun.hh	487
tgbaalgos/reductgba_sim.hh	487
tgbaalgos/replayrun.hh	489
tgbaalgos/rundotdec.hh	489
tgbaalgos/save.hh	423
tgbaalgos/se05.hh	490
tgbaalgos/stats.hh	490
tgbaalgos/tau03.hh	491
tgbaalgos/tau03opt.hh	491
tgbaalgos/weight.hh	491
tgbaalgos/gtec/ce.hh	476
tgbaalgos/gtec/explscs.hh	477
tgbaalgos/gtec/gtec.hh	478
tgbaalgos/gtec/nsheap.hh	479
tgbaalgos/gtec/scsstack.hh	480
tgbaalgos/gtec/status.hh	481
tgbaalgos/public.hh	427

## 8 spot Module Documentation

### 8.1 LTL formulae

#### Modules

- [Essential LTL types](#)
- [LTL Abstract Syntax Tree](#)
- [LTL environments](#)
- [Algorithms for LTL formulae](#)

#### 8.1.1 Detailed Description

This module gathers types and definitions related to LTL formulae.

### 8.2 TGBA (Transition-based Generalized Büchi Automata)

#### Modules

- [Essential TGBA types](#)
- [TGBA representations](#)
- [TGBA algorithms](#)

#### 8.2.1 Detailed Description

Spot is centered around the [spot::tgba](#) type. This type and its cousins are listed [here](#). This is an abstract interface. Its implementations are either [concrete representations](#), or [on-the-fly algorithms](#). Other algorithms that work on [spot::tgba](#) are [listed separately](#).

### 8.3 Emptiness-check algorithms for SSP

#### Functions

- `couvreur99_check * spot::couvreur99\_check\_ssp\_semi (const tgba *ssp_automata)`
- `couvreur99_check * spot::couvreur99\_check\_ssp\_shy\_semi (const tgba *ssp_automata)`
- `couvreur99_check * spot::couvreur99\_check\_ssp\_shy (const tgba *ssp_automata)`

#### 8.3.1 Function Documentation

**8.3.1.1** `couvreur99_check* couvreur99_check_ssp_semi (const tgba * ssp_automata)`

**8.3.1.2** `couvreur99_check* couvreur99_check_ssp_shy (const tgba * ssp_automata)`

**8.3.1.3** `couvreur99_check* couvreur99_check_ssp_shy_semi (const tgba * ssp_automata)`

## 8.4 Essential LTL types

### Classes

- class `spot::ltl::formula`  
*An LTL formula.*
- struct `spot::ltl::visitor`  
*Formula visitor that can modify the formula.*
- class `spot::ltl::environment`  
*An environment that describes atomic propositions.*

### Functions

- formula \* `spot::ltl::clone` (const formula \*f)  
*Clone a formula.*
- void `spot::ltl::destroy` (const formula \*f)  
*Destroys a formula.*

#### 8.4.1 Function Documentation

##### 8.4.1.1 formula\* clone (const formula \*f)

Clone a formula.

##### 8.4.1.2 void destroy (const formula \*f)

Destroys a formula.

## 8.5 LTL Abstract Syntax Tree

### Classes

- class `spot::ltl::atomic_prop`  
*Atomic propositions.*
- class `spot::ltl::binop`  
*Binary operator.*
- class `spot::ltl::constant`  
*A constant (True or False).*
- class `spot::ltl::formula`  
*An LTL formula.*
- class `spot::ltl::multop`

*Multi-operand operators.*

- class `spot::ltl::ref_formula`  
*A reference-counted LTL formula.*
- class `spot::ltl::unop`  
*Unary operators.*

## 8.6 LTL environments

### Classes

- class `spot::ltl::declarative_environment`  
*A declarative environment.*
- class `spot::ltl::default_environment`  
*A laxist environment.*

### 8.6.1 Detailed Description

LTL environment implementations.

## 8.7 Algorithms for LTL formulae

### Modules

- `Input/Output of LTL formulae`
- `Derivable visitors`
- `Rewriting LTL formulae`
- `Miscellaneous algorithms for LTL formulae`

## 8.8 Input/Output of LTL formulae

### Classes

- class `spot::ltl::random_ltl`  
*Generate random LTL formulae.*

### Typedefs

- typedef `std::pair< yy::Location, std::string >` `spot::ltl::parse_error`  
*A parse diagnostic with its location.*
- typedef `std::list< parse_error >` `spot::ltl::parse_error_list`  
*A list of parser diagnostics, as filled by parse.*

## Functions

- formula \* `spot::ltl::parse` (const std::string &ltl\_string, `parse_error_list` &error\_list, environment &env=default\_environment::instance(), bool debug=false)  
*Build a formula from an LTL string.*
- bool `spot::ltl::format_parse_errors` (std::ostream &os, const std::string &ltl\_string, `parse_error_list` &error\_list)  
*Format diagnostics produced by `spot::ltl::parse`.*
- std::ostream & `spot::ltl::dotty` (std::ostream &os, const formula \*f)  
*Write a formula tree using dot's syntax.*
- std::ostream & `spot::ltl::dump` (std::ostream &os, const formula \*f)  
*Dump a formula tree.*
- std::ostream & `spot::ltl::to_string` (const formula \*f, std::ostream &os)  
*Output a formula as a (parsable) string.*
- std::string `spot::ltl::to_string` (const formula \*f)  
*Convert a formula into a (parsable) string.*
- std::ostream & `spot::ltl::to_spin_string` (const formula \*f, std::ostream &os)  
*Output a formula as a (parsable by Spin) string.*
- std::string `spot::ltl::to_spin_string` (const formula \*f)  
*Convert a formula into a (parsable by Spin) string.*

### 8.8.1 Typedef Documentation

#### 8.8.1.1 typedef std::pair<yy::Location, std::string> `spot::ltl::parse_error`

A parse diagnostic with its location.

#### 8.8.1.2 typedef std::list<`parse_error`> `spot::ltl::parse_error_list`

A list of parser diagnostics, as filled by parse.

### 8.8.2 Function Documentation

#### 8.8.2.1 std::ostream& `dotty` (std::ostream &os, const formula \*f)

Write a formula tree using dot's syntax.

#### Parameters:

- os* The stream where it should be output.
- f* The formula to translate.

dot is part of the GraphViz package <http://www.research.att.com/sw/tools/graphviz/>

**8.8.2.2 `std::ostream& dump (std::ostream & os, const formula *f)`**

Dump a formula tree.

**Parameters:**

- os* The stream where it should be output.
- f* The formula to dump.

This is useful to display a formula when debugging.

**8.8.2.3 `bool format_parse_errors (std::ostream & os, const std::string & ltl_string, parse_error_list & error_list)`**

Format diagnostics produced by `spot::ltl::parse`.

**Parameters:**

- os* Where diagnostics should be output.
- ltl\_string* The string that were parsed.
- error\_list* The error list filled by `spot::ltl::parse` while parsing *ltl\_string*.

**Returns:**

- `true` iff any diagnostic was output.

**8.8.2.4 `formula* parse (const std::string & ltl_string, parse_error_list & error_list, environment & env = default_environment::instance(), bool debug = false)`**

Build a formula from an LTL string.

**Parameters:**

- ltl\_string* The string to parse.
- error\_list* A list that will be filled with parse errors that occurred during parsing.
- env* The environment into which parsing should take place.
- debug* When true, causes the parser to trace its execution.

**Returns:**

- A pointer to the formula built from *ltl\_string*, or 0 if the input was unparsable.

Note that the parser usually tries to recover from errors. It can return a non zero value even if it encountered an error during the parsing of *ltl\_string*. If you want to make sure *ltl\_string* was parsed successfully, check *error\_list* for emptiness.

**Warning:**

- This function is not reentrant.

**8.8.2.5 `std::string to_spin_string (const formula *f)`**

Convert a formula into a (parsable by Spin) string.

**Parameters:**

- f* The formula to translate.

**8.8.2.6 `std::ostream& to_spin_string (const formula *f, std::ostream & os)`**

Output a formula as a (parsable by Spin) string.

**Parameters:**

- f* The formula to translate.
- os* The stream where it should be output.

**8.8.2.7 `std::string to_string (const formula *f)`**

Convert a formula into a (parsable) string.

**Parameters:**

- f* The formula to translate.

**8.8.2.8 `std::ostream& to_string (const formula *f, std::ostream & os)`**

Output a formula as a (parsable) string.

**Parameters:**

- f* The formula to translate.
- os* The stream where it should be output.

**8.9 Derivable visitors****Classes**

- class `spot::ltl::clone_visitor`  
*Clone a formula.*
- class `spot::ltl::unabbreviate_logic_visitor`  
*Clone and rewrite a formula to remove most of the abbreviated logical operators.*
- class `spot::ltl::postfix_visitor`  
*Apply an algorithm on each node of an AST, during a postfix traversal.*
- class `spot::ltl::simplify_fg_visitor`  
*Replace `true ∪ f` and `false R g` by `F f` and `G g`.*
- class `spot::ltl::unabbreviate_ltl_visitor`  
*Clone and rewrite a formula to remove most of the abbreviated LTL and logical operators.*

**8.10 Rewriting LTL formulae****Enumerations**

- enum `spot::ltl::reduce_options` {  
`spot::ltl::Reduce_None` = 0, `spot::ltl::Reduce_Basics` = 1, `spot::ltl::Reduce_Syntactic_Implications`  
= 2, `spot::ltl::Reduce_Eventuality_And_Universality` = 4,  
`spot::ltl::Reduce_All` = -1U }

Options for `spot::ltl::reduce`.

## Functions

- formula \* `spot::ltl::basic_reduce` (const formula \*f)  
*Basic rewritings.*
- formula \* `spot::ltl::unabbreviate_logic` (const formula \*f)  
*Clone and rewrite a formula to remove most of the abbreviated logical operators.*
- formula \* `spot::ltl::negative_normal_form` (const formula \*f, bool negated=false)  
*Build the negative normal form of f.*
- formula \* `spot::ltl::reduce` (const formula \*f, int opt=Reduce\_All)  
*Reduce a formula f.*
- formula \* `spot::ltl::simplify_f_g` (const formula \*f)  
*Replace true  $\cup$  f and false  $\cap$  g by F f and G g.*

### 8.10.1 Enumeration Type Documentation

#### 8.10.1.1 enum `reduce_options`

Options for `spot::ltl::reduce`.

#### Enumeration values:

***Reduce\_None*** No reduction.

***Reduce\_Basics*** Basic reductions.

***Reduce\_Syntactic\_Implications*** Somenzi & Bloem syntactic implication.

***Reduce\_Eventuality\_And\_Universality*** Etessami & Holzmann eventuality and universality reductions.

***Reduce\_All*** All reductions.

### 8.10.2 Function Documentation

#### 8.10.2.1 formula\* `basic_reduce` (const formula \*f)

Basic rewritings.

#### 8.10.2.2 formula\* `negative_normal_form` (const formula \*f, bool *negated* = false)

Build the negative normal form of f.

All negations of the formula are pushed in front of the atomic propositions.

#### Parameters:

*f* The formula to normalize.

***negated*** If true, return the negative normal form of  $\neg f$



Note that this will not remove abbreviated operators. If you want to remove abbreviations, call `spot::ltl::unabbreviate_logic` or `spot::ltl::unabbreviate_ltl` first. (Calling these functions after `spot::ltl::negative_normal_form` would likely produce a formula which is not in negative normal form.)

### 8.10.2.3 formula\* reduce (const formula \*f, int opt = Reduce\_All)

Reduce a formula *f*.

#### Parameters:

*f* the formula to reduce

*opt* a conjunction of `spot::ltl::reduce_options` specifying which optimizations to apply.

#### Returns:

the reduced formula

### 8.10.2.4 formula\* simplify\_f\_g (const formula \*f)

Replace `true U f` and `false R g` by `F f` and `G g`.

### 8.10.2.5 formula\* unabbreviate\_logic (const formula \*f)

Clone and rewrite a formula to remove most of the abbreviated logical operators.

This will rewrite binary operators such as `binop::Implies`, `binop::Equals`, and `binop::Xor`, using only `unop::Not`, `multop::Or`, and `multop::And`.

## 8.11 Miscellaneous algorithms for LTL formulae

### Typedefs

- `typedef std::set< atomic_prop *, formula_ptr_less_than > spot::ltl::atomic_prop_set`  
*Set of atomic propositions.*

### Functions

- `atomic_prop_set * spot::ltl::atomic_prop_collect (const formula *f, atomic_prop_set *s=0)`  
*Return the set of atomic propositions occurring in a formula.*
- `bool spot::ltl::is_GF (const formula *f)`  
*Whether a formula starts with GF.*
- `bool spot::ltl::is_FG (const formula *f)`  
*Whether a formula starts with FG.*
- `int spot::ltl::length (const formula *f)`  
*Compute the length of a formula.*
- `bool spot::ltl::is_eventual (const formula *f)`

*Check whether a formula is a pure eventuality.*

- bool `spot::ltl::is_universal` (const formula \*f)  
*Check whether a formula is purely universal.*
- bool `spot::ltl::syntactic_implication` (const formula \*f1, const formula \*f2)  
*Syntactic implication.*
- bool `spot::ltl::syntactic_implication_neg` (const formula \*f1, const formula \*f2, bool right)  
*Syntactic implication.*

### 8.11.1 Typedef Documentation

#### 8.11.1.1 typedef std::set<atomic\_prop\*, formula\_ptr\_less\_than> `spot::ltl::atomic_prop_set`

Set of atomic propositions.

### 8.11.2 Function Documentation

#### 8.11.2.1 `atomic_prop_set* atomic_prop_collect` (const formula \*f, `atomic_prop_set` \*s = 0)

Return the set of atomic propositions occurring in a formula.

##### Parameters:

*f* the formula to inspect

*s* an existing set to fill with atomic\_propositions discovered, or 0 if the set should be allocated by the function.

##### Returns:

A pointer to the supplied set, *s*, augmented with atomic propositions occurring in *f*; or a newly allocated set containing all these atomic propositions if *s* is 0.

#### 8.11.2.2 bool `is_eventual` (const formula \*f)

Check whether a formula is a pure eventuality.

Pure eventuality formulae are defined in

```
@InProceedings{ etessami.00.concur,
  author   = {Kousha Etessami and Gerard J. Holzmann},
  title    = {Optimizing {B\"u}chi Automata},
  booktitle = {Proceedings of the 11th International Conference on
    Concurrency Theory (Concur'2000)},
  pages    = {153--167},
  year     = {2000},
  editor   = {C. Palamidessi},
  volume   = {1877},
  series   = {Lecture Notes in Computer Science},
  publisher = {Springer-Verlag}
}
```

A word that satisfies a pure eventuality can be prefixed by anything and still satisfies the formula.

**8.11.2.3 bool is\_FG (const formula \*f)**

Whether a formula starts with FG.

**8.11.2.4 bool is\_GF (const formula \*f)**

Whether a formula starts with GF.

**8.11.2.5 bool is\_universal (const formula \*f)**

Check whether a formula is purely universal.

Purely universal formulae are defined in

```
@InProceedings{ etessami.00.concur,
  author   = {Kousha Etessami and Gerard J. Holzmann},
  title    = {Optimizing {B\"u}chi Automata},
  booktitle = {Proceedings of the 11th International Conference on
    Concurrency Theory (Concur'2000)},
  pages    = {153--167},
  year     = {2000},
  editor   = {C. Palamidessi},
  volume   = {1877},
  series   = {Lecture Notes in Computer Science},
  publisher = {Springer-Verlag}
}
```

Any (non-empty) suffix of a word that satisfies if purely universal formula also satisfies the formula.

**8.11.2.6 int length (const formula \*f)**

Compute the length of a formula.

The length of a formula is the number of atomic properties, constants, and operators (logical and temporal) occurring in the formula. `spot::ltl::multops` count only for 1, even if they have more than two operands (e.g. `a | b | c` has length 4, because `|` is represented once internally).

**8.11.2.7 bool syntactic\_implication (const formula \*f1, const formula \*f2)**

Syntactic implication.

This comes from

```
@InProceedings{ somenzi.00.cav,
  author   = {Fabio Somenzi and Roderick Bloem},
  title    = {Efficient {B\"u}chi Automata for {LTL} Formulae},
  booktitle = {Proceedings of the 12th International Conference on
    Computer Aided Verification (CAV'00)},
  pages    = {247--263},
  year     = {2000},
  volume   = {1855},
  series   = {Lecture Notes in Computer Science},
  publisher = {Springer-Verlag}
}
```

**8.11.2.8 bool syntactic\_implication\_neg (const formula \*f1, const formula \*f2, bool right)**

Syntactic implication.

If right==false, true if !f1 < f2, false otherwise. If right==true, true if f1 < !f2, false otherwise.

See also:

[syntactic\\_implication](#)

**8.12 Miscellaneous helper algorithms**

Whether a word is bare.

**Modules**

- [Hashing functions](#)
- [Random functions](#)

**Classes**

- class [spot::bdd\\_allocator](#)  
*Manage ranges of variables.*
- struct [spot::bdd\\_less\\_than](#)  
*Comparison functor for BDDs.*
- class [spot::free\\_list](#)  
*Manage list of free integers.*
- class [spot::minato\\_isop](#)  
*Generate an irredundant sum-of-products (ISOP) form of a BDD function.*
- class [spot::loopless\\_modular\\_mixed\\_radix\\_gray\\_code](#)  
*Loopless modular mixed radix Gray code iteration.*
- struct [spot::time\\_info](#)  
*A structure to record elapsed time in clock ticks.*
- class [spot::timer](#)  
*A timekeeper that accumulate interval of time.*
- class [spot::timer\\_map](#)  
*A map of timer, where each timer has a name.*

**Functions**

- bool [spot::is\\_bare\\_word](#) (const char \*str)
- std::string [spot::quote\\_unless\\_bare\\_word](#) (const std::string &str)  
*Double-quote words that are not bare.*

- `std::ostream & spot::escape_str (std::ostream &os, const std::string &str)`  
*Escape " and \ characters in str.*
- `std::string spot::escape_str (const std::string &str)`  
*Escape " and \ characters in str.*
- `const char * spot::version ()`  
*Return Spot's version.*

### 8.12.1 Detailed Description

Whether a word is bare.

Bare words should start with a letter or an underscore, and consist solely of alphanumeric characters and underscores.

### 8.12.2 Function Documentation

#### 8.12.2.1 `std::string escape_str (const std::string & str)`

Escape " and \ characters in *str*.

#### 8.12.2.2 `std::ostream& escape_str (std::ostream & os, const std::string & str)`

Escape " and \ characters in *str*.

#### 8.12.2.3 `bool is_bare_word (const char * str)`

#### 8.12.2.4 `std::string quote_unless_bare_word (const std::string & str)`

Double-quote words that are not bare.

See also:

[is\\_bare\\_word](#)

#### 8.12.2.5 `const char* version ()`

Return Spot's version.

## 8.13 Hashing functions

### Classes

- struct [spot::ltd::formula\\_ptr\\_hash](#)  
*Hash Function for const formula\*.*
- struct [spot::ptr\\_hash< T >](#)

*A hash function for pointers.*

- struct `spot::string_hash`  
*A hash function for strings.*
- struct `spot::state_ptr_hash`  
*Hash Function for state\*.*

## Functions

- `size_t spot::wang32_hash (size_t key)`  
*Thomas Wang's 32 bit hash function.*

### 8.13.1 Function Documentation

#### 8.13.1.1 `size_t wang32_hash (size_t key) [inline]`

Thomas Wang's 32 bit hash function.

Hash an integer amongst the integers. <http://www.concentric.net/~Ttwang/tech/inthash.htm>

## 8.14 Random functions

### Classes

- class `spot::barand< gen >`  
*Compute pseudo-random integer value between 0 and n included, following a binomial distribution for probability p.*

### Functions

- void `spot::srand (unsigned int seed)`  
*Reset the seed of the pseudo-random number generator.*
- int `spot::rrand (int min, int max)`  
*Compute a pseudo-random integer value between min and max included.*
- int `spot::mrand (int max)`  
*Compute a pseudo-random integer value between 0 and max-1 included.*
- double `spot::drand ()`  
*Compute a pseudo-random double value between 0.0 and 1.0 (1.0 excluded).*
- double `spot::nrand ()`  
*Compute a pseudo-random double value following a standard normal distribution. (Odeh & Evans).*

- double `spot::bmrnd ()`  
*Compute a pseudo-random double value following a standard normal distribution. (Box-Muller).*
- int `spot::prand (double p)`  
*Return a pseudo-random positive integer value following a Poisson distribution with parameter  $p$ .*

### 8.14.1 Function Documentation

#### 8.14.1.1 double `bmrnd ()`

Compute a pseudo-random double value following a standard normal distribution. (Box-Muller).

This uses the polar form of the Box-Muller transform to generate random values.

#### 8.14.1.2 double `drand ()`

Compute a pseudo-random double value between 0.0 and 1.0 (1.0 excluded).

See also:

[mrand](#), [rrand](#), [srand](#)

#### 8.14.1.3 int `mrand (int max)`

Compute a pseudo-random integer value between 0 and  $max-1$  included.

See also:

[drand](#), [rrand](#), [srand](#)

#### 8.14.1.4 double `nrnd ()`

Compute a pseudo-random double value following a standard normal distribution. (Odeh & Evans).

This uses a polynomial approximation of the inverse cumulated density function from Odeh & Evans, Journal of Applied Statistics, 1974, vol 23, pp 96-97.

#### 8.14.1.5 int `prand (double p)`

Return a pseudo-random positive integer value following a Poisson distribution with parameter  $p$ .

**Precondition:**

$p > 0$

#### 8.14.1.6 int `rrand (int min, int max)`

Compute a pseudo-random integer value between  $min$  and  $max$  included.

See also:

[drand](#), [mrand](#), [srand](#)

#### 8.14.1.7 void srand (unsigned int seed)

Reset the seed of the pseudo-random number generator.

See also:

[drand](#), [mrnd](#), [rrand](#)

## 8.15 Essential TGBA types

### Classes

- class [spot::bdd\\_dict](#)
- class [spot::state](#)  
*Abstract class for states.*
- struct [spot::state\\_ptr\\_less\\_than](#)  
*Strict Weak Ordering for state\*.*
- struct [spot::state\\_ptr\\_equal](#)  
*An Equivalence Relation for state\*.*
- struct [spot::state\\_ptr\\_hash](#)  
*Hash Function for state\*.*
- class [spot::tgba\\_succ\\_iterator](#)  
*Iterate over the successors of a state.*
- class [spot::tgba](#)  
*A Transition-based Generalized Büchi Automaton.*

## 8.16 TGBA representations

### Classes

- class [spot::state\\_bdd](#)
- class [spot::tgba\\_succ\\_iterator\\_concrete](#)
- class [spot::tgba\\_bdd\\_concrete](#)  
*A concrete [spot::tgba](#) implemented using BDDs.*
- class [spot::tgba\\_explicit](#)
- class [spot::state\\_explicit](#)
- class [spot::tgba\\_explicit\\_succ\\_iterator](#)
- class [spot::tgba\\_reduc](#)



## 8.17 TGBA algorithms

### Modules

- [TGBA on-the-fly algorithms](#)
- [Input/Output of TGBA](#)
- [Translating LTL formulae into TGBA](#)
- [Algorithm patterns](#)
- [TGBA simplifications](#)
- [Miscellaneous algorithms on TGBA](#)
- [Emptiness-checks](#)

### Functions

- `tgba_bdd_concrete * spot::product (const tgba_bdd_concrete *left, const tgba_bdd_concrete *right)`

*Multiplies two [spot::tgba\\_bdd\\_concrete](#) automata.*

#### 8.17.1 Function Documentation

##### 8.17.1.1 `tgba_bdd_concrete* product (const tgba_bdd_concrete * left, const tgba_bdd_concrete * right)`

Multiplies two [spot::tgba\\_bdd\\_concrete](#) automata.

This function builds the resulting product as another [spot::tgba\\_bdd\\_concrete](#) automaton.

## 8.18 TGBA on-the-fly algorithms

### Classes

- class [spot::state\\_product](#)  
*A state for [spot::tgba\\_product](#).*
- class [spot::tgba\\_tba\\_proxy](#)  
*Degeneralize a [spot::tgba](#) on the fly, producing a TBA.*
- class [spot::tgba\\_sba\\_proxy](#)  
*Degeneralize a [spot::tgba](#) on the fly, producing an SBA.*

## 8.19 Input/Output of TGBA

### Modules

- [Decorating the dot output](#)

## Typedefs

- `typedef std::pair< yy::Location, std::string > spot::tgba\_parse\_error`  
A parse diagnostic with its location.
- `typedef std::list< tgba\_parse\_error > spot::tgba\_parse\_error\_list`  
A list of parser diagnostics, as filled by parse.

## Functions

- `std::ostream & spot::dotty\_reachable (std::ostream &os, const tgba *g, dotty_decorator *dd=dotty_decorator::instance())`  
Print reachable states in dot format.
- `std::ostream & spot::lbt\_reachable (std::ostream &os, const tgba *g)`  
Print reachable states in LBT format.
- `std::ostream & spot::never\_claim\_reachable (std::ostream &os, const tgba_sba_proxy *g, const ltl::formula *f=0)`  
Print reachable states in Spin never claim format.
- `std::ostream & spot::tgba\_save\_reachable (std::ostream &os, const tgba *g)`  
Save reachable states in text format.
- `tgba_explicit * spot::tgba\_parse (const std::string &filename, tgba\_parse\_error\_list &error_list, bdd_dict *dict, ltl::environment &env=ltl::default_environment::instance(), bool debug=false)`  
Build a [spot::tgba\\_explicit](#) from a text file.
- `bool spot::format\_tgba\_parse\_errors (std::ostream &os, const std::string &filename, tgba\_parse\_error\_list &error_list)`  
Format diagnostics produced by [spot::tgba\\_parse](#).

### 8.19.1 Typedef Documentation

#### 8.19.1.1 `typedef std::pair<yy::Location, std::string> spot::tgba\_parse\_error`

A parse diagnostic with its location.

#### 8.19.1.2 `typedef std::list<tgba\_parse\_error> spot::tgba\_parse\_error\_list`

A list of parser diagnostics, as filled by parse.

### 8.19.2 Function Documentation

#### 8.19.2.1 `std::ostream& dotty\_reachable (std::ostream &os, const tgba *g, dotty_decorator *dd = dotty_decorator::instance())`

Print reachable states in dot format.

The *dd* argument allows to customize the output in various ways. See [this page](#) for a list of available decorators.

### 8.19.2.2 `bool format_tgba_parse_errors (std::ostream & os, const std::string & filename, tgba\_parse\_error\_list & error_list)`

Format diagnostics produced by [spot::tgba\\_parse](#).

#### Parameters:

- os* Where diagnostics should be output.
- filename* The filename that should appear in the diagnostics.
- error\_list* The error list filled by [spot::ltl::parse](#) while parsing *ltl\_string*.

#### Returns:

`true` iff any diagnostic was output.

### 8.19.2.3 `std::ostream& lbtt_reachable (std::ostream & os, const tgba * g)`

Print reachable states in LBTT format.

#### Parameters:

- g* The automata to print.
- os* Where to print.

### 8.19.2.4 `std::ostream& never_claim_reachable (std::ostream & os, const tgba_sba_proxy * g, const ltl::formula * f = 0)`

Print reachable states in Spin never claim format.

#### Parameters:

- os* The output stream to print on.
- g* The degeneralized automaton to output.
- f* The (optional) formula associated to the automaton. If given it will be output as a comment.

### 8.19.2.5 `tgba_explicit* tgba_parse (const std::string & filename, tgba\_parse\_error\_list & error_list, bdd_dict * dict, ltl::environment & env = ltl::default_environment::instance(), bool debug = false)`

Build a [spot::tgba\\_explicit](#) from a text file.

#### Parameters:

- filename* The name of the file to parse.
- error\_list* A list that will be filled with parse errors that occurred during parsing.
- dict* The BDD dictionary where to use.
- env* The environment into which parsing should take place.
- debug* When true, causes the parser to trace its execution.

#### Returns:

A pointer to the tgba built from *filename*, or 0 if the file could not be opened.

Note that the parser usually tries to recover from errors. It can return a non zero value even if it encountered error during the parsing of *filename*. If you want to make sure *filename* was parsed successfully, check *error\_list* for emptiness.

#### Warning:

This function is not reentrant.

### 8.19.2.6 `std::ostream& tgba_save_reachable (std::ostream & os, const tgba * g)`

Save reachable states in text format.

## 8.20 Translating LTL formulae into TGBA

### Functions

- `tgba_explicit * spot::ltl_to_tgba_fm` (`const ltl::formula *f`, `bdd_dict *dict`, `bool exprop=false`, `bool symb_merge=true`, `bool branching_postponement=false`, `bool fair_loop_approx=false`, `const ltl::atomic_prop_set *unobs=0`)

*Build a `spot::tgba_explicit*` from an LTL formula.*

- `tgba_bdd_concrete * spot::ltl_to_tgba_lacim` (`const ltl::formula *f`, `bdd_dict *dict`)

*Build a `spot::tgba_bdd_concrete` from an LTL formula.*

### 8.20.1 Function Documentation

**8.20.1.1 `tgba_explicit* ltl_to_tgba_fm`** (`const ltl::formula *f`, `bdd_dict *dict`, `bool exprop = false`, `bool symb_merge = true`, `bool branching_postponement = false`, `bool fair_loop_approx = false`, `const ltl::atomic_prop_set *unobs = 0`)

Build a `spot::tgba_explicit*` from an LTL formula.

This is based on the following paper.

```
@InProceedings{couvreur.99.fm,
  author    = {Jean-Michel Couvreur},
  title     = {On-the-fly Verification of Temporal Logic},
  pages     = {253--271},
  editor    = {Jeannette M. Wing and Jim Woodcock and Jim Davies},
  booktitle = {Proceedings of the World Congress on Formal Methods in the
    Development of Computing Systems (FM'99)},
  publisher = {Springer-Verlag},
  series    = {Lecture Notes in Computer Science},
  volume    = {1708},
  year      = {1999},
  address   = {Toulouse, France},
  month     = {September},
  isbn      = {3-540-66587-0}
}
```

### Parameters:

***f*** The formula to translate into an automaton.

***dict*** The `spot::bdd_dict` the constructed automata should use.

***exprop*** When set, the algorithm will consider all properties combinations possible on each state, in an attempt to reduce the non-determinism. The automaton will have the same size as without this option, but because the transition will be more deterministic, the product automaton will be smaller (or, at worse, equal).

***symb\_merge*** When false, states with the same symbolic representation (these are equivalent formulae) will not be merged.

***branching\_postponement*** When set, several transitions leaving from the same state with the same label (i.e., condition + acceptance conditions) will be merged. This correspond to an optimization described in the following paper.

```

@InProceedings{    sebastiani.03.charme,
  author   = {Roberto Sebastiani and Stefano Tonetta},
  title    = {"More Deterministic" vs. "Smaller" B{"u"}chi Automata for
    Efficient LTL Model Checking},
  booktitle = {Proceedings for the 12th Advanced Research Working
    Conference on Correct Hardware Design and Verification
    Methods (CHARME'03)},
  pages     = {126--140},
  year      = {2003},
  editor    = {G. Goos and J. Hartmanis and J. van Leeuwen},
  volume    = {2860},
  series    = {Lectures Notes in Computer Science},
  month     = {October},
  publisher = {Springer-Verlag}
}

```

***fair\_loop\_approx*** When set, a really simple characterization of unstable state is used to suppress all acceptance conditions from incoming transitions.

***unobs*** When non-zero, the atomic propositions in the LTL formula are interpreted as events that exclude each other. The events in the formula are observable events, and *unobs* can be filled with additional unobservable events.

**Returns:**

A [spot::tgba\\_explicit](#) that recognizes the language of *f*.

### 8.20.1.2 tgba\_bdd\_concrete\* ltl\_to\_tgba\_lacim (const ltl::formula \*f, bdd\_dict \*dict)

Build a [spot::tgba\\_bdd\\_concrete](#) from an LTL formula.

This is based on the following paper.

```

@InProceedings{    couvreur.00.lacim,
  author          = {Jean-Michel Couvreur},
  title           = {Un point de vue symbolique sur la logique temporelle
    lin{"'e"}aire},
  booktitle       = {Actes du Colloque LaCIM 2000},
  month           = {August},
  year            = {2000},
  pages           = {131--140},
  volume          = {27},
  series          = {Publications du LaCIM},
  publisher        = {Universit{"'e"} du Qu{"'e"}bec {"'a"} Montr{"'e"}al},
  editor          = {Pierre Leroux}
}

```

**Parameters:**

*f* The formula to translate into an automaton.

*dict* The [spot::bdd\\_dict](#) the constructed automata should use.

**Returns:**

A [spot::tgba\\_bdd\\_concrete](#) that recognizes the language of *f*.

## 8.21 Algorithm patterns

### Classes

- class [spot::tgba\\_reachable\\_iterator](#)

*Iterate over all reachable states of a `spot::tgba`.*

- class `spot::tgba_reachable_iterator_depth_first`  
*An implementation of `spot::tgba_reachable_iterator` that browses states depth first.*
- class `spot::tgba_reachable_iterator_breadth_first`  
*An implementation of `spot::tgba_reachable_iterator` that browses states breadth first.*

## 8.22 TGBA simplifications

### Classes

- class `spot::parity_game_graph`  
*Parity game graph which compute a simulation relation.*
- class `spot::spoiler_node`  
*Spoiler node of parity game graph.*
- class `spot::duplicator_node`  
*Duplicator node of parity game graph.*
- class `spot::parity_game_graph_direct`  
*Parity game graph which compute the direct simulation relation.*
- class `spot::spoiler_node_delayed`  
*Spoiler node of parity game graph for delayed simulation.*
- class `spot::duplicator_node_delayed`  
*Duplicator node of parity game graph for delayed simulation.*
- class `spot::parity_game_graph_delayed`

### Typedefs

- typedef `Sgi::vector< spoiler_node * >` `spot::sn_v`
- typedef `Sgi::vector< duplicator_node * >` `spot::dn_v`
- typedef `Sgi::vector< const state * >` `spot::s_v`

### Enumerations

- enum `spot::reduce_tgba_options` {  
`spot::Reduce_None = 0`, `spot::Reduce_quotient_Dir_Sim = 1`, `spot::Reduce_transition_Dir_Sim = 2`,  
`spot::Reduce_quotient_Del_Sim = 4`,  
`spot::Reduce_transition_Del_Sim = 8`, `spot::Reduce_Scc = 16`, `spot::Reduce_All = -1U` }  
*Options for reduce.*

## Functions

- `tgba * spot::reduc_tgba_sim` (const `tgba *a`, int `opt=Reduce_All`)  
*Remove some node of the automata using a simulation relation.*
- `direct_simulation_relation * spot::get_direct_relation_simulation` (const `tgba *a`, `std::ostream &os`, int `opt=-1`)  
*Compute a direct simulation relation on state of `tgba f`.*
- `delayed_simulation_relation * spot::get_delayed_relation_simulation` (const `tgba *a`, `std::ostream &os`, int `opt=-1`)  
*Compute a delayed simulation relation on state of `tgba f`.*
- `void spot::free_relation_simulation` (`direct_simulation_relation *rel`)  
*To free a simulation relation.*
- `void spot::free_relation_simulation` (`delayed_simulation_relation *rel`)  
*To free a simulation relation.*

### 8.22.1 Typedef Documentation

8.22.1.1 `typedef Sgi::vector<duplicator_node*> spot::dn_v`

8.22.1.2 `typedef Sgi::vector<const state*> spot::s_v`

8.22.1.3 `typedef Sgi::vector<spoiler_node*> spot::sn_v`

### 8.22.2 Enumeration Type Documentation

8.22.2.1 `enum reduce_tgba_options`

Options for reduce.

#### Enumeration values:

*Reduce\_None* No reduction.

*Reduce\_quotient\_Dir\_Sim* Reduction of state using direct simulation relation.

*Reduce\_transition\_Dir\_Sim* Reduction of transitions using direct simulation relation.

*Reduce\_quotient\_Del\_Sim* Reduction of state using delayed simulation relation.

*Reduce\_transition\_Del\_Sim* Reduction of transition using delayed simulation relation.

*Reduce\_Scc* Reduction using SCC.

*Reduce\_All* All reductions.

### 8.22.3 Function Documentation

8.22.3.1 `void free_relation_simulation` (`delayed_simulation_relation * rel`)

To free a simulation relation.

**8.22.3.2 void free\_relation\_simulation (direct\_simulation\_relation \* rel)**

To free a simulation relation.

**8.22.3.3 delayed\_simulation\_relation\* get\_delayed\_relation\_simulation (const tgba \* a, std::ostream & os, int opt = -1)**

Compute a delayed simulation relation on state of tgba *f*.

**8.22.3.4 direct\_simulation\_relation\* get\_direct\_relation\_simulation (const tgba \* a, std::ostream & os, int opt = -1)**

Compute a direct simulation relation on state of tgba *f*.

**8.22.3.5 tgba\* reduc\_tgba\_sim (const tgba \* a, int opt = Reduce\_All)**

Remove some node of the automata using a simulation relation.

**Parameters:**

*a* the automata to reduce.

*opt* a conjunction of [spot::reduce\\_tgba\\_options](#) specifying which optimizations to apply.

**Returns:**

the reduced automata.

**8.23 Miscellaneous algorithms on TGBA****Classes**

- class [spot::bfs\\_steps](#)  
*Make a BFS in a [spot::tgba](#) to compute a [tgba\\_run::steps](#).*
- struct [spot::tgba\\_statistics](#)

**Functions**

- tgba\_explicit \* [spot::tgba\\_dupexp\\_bfs](#) (const tgba \* aut)  
*Build an explicit automata from all states of aut, numbering states in bread first order as they are processed.*
- tgba\_explicit \* [spot::tgba\\_dupexp\\_dfs](#) (const tgba \* aut)  
*Build an explicit automata from all states of aut, numbering states in depth first order as they are processed.*
- tgba\_explicit \* [spot::tgba\\_powerset](#) (const tgba \* aut)  
*Build a deterministic automaton, ignoring acceptance conditions.*
- tgba \* [spot::random\\_graph](#) (int n, float d, const [ltl::atomic\\_prop\\_set](#) \* ap, bdd\_dict \* dict, int n\_acc=0, float a=0.1, float t=0.5, [ltl::environment](#) \* env=&[ltl::default\\_environment::instance\(\)](#))  
*Construct a tgba randomly.*
- tgba\_statistics [spot::stats\\_reachable](#) (const tgba \* g)  
*Compute statistics for an automaton.*



### 8.23.1 Function Documentation

**8.23.1.1** `tgba* random_graph (int n, float d, const ltl::atomic\_prop\_set * ap, bdd_dict * dict, int n_acc = 0, float a = 0.1, float t = 0.5, ltl::environment * env = &ltltl::default_environment::instance())`

Construct a tgba randomly.

**Parameters:**

- n* The number of states wanted in the automata. All states will be connected, and there will be no dead state.
- d* The density of the automata. This is the probability (between 0.0 and 1.0), to add a transition between two states. All states have at least one outgoing transition, so *d* is considered only when adding the remaining transition. A density of 1 means all states will be connected to each other.
- ap* The list of atomic property that should label the transition.
- dict* The [bdd\\_dict](#) to used for this automata.
- n\_acc* The number of acceptance sets to use.
- a* The probability (between 0.0 and 1.0) that a transition belongs to an acceptance set.
- t* The probability (between 0.0 and 1.0) that an atomic proposition is true.
- env* The environment in which to declare the acceptance conditions.

This algorithms is adapted from the one in Fig 6.2 page 48 of

```
@TechReport{ tauriainen.00.a66,
  author = {Heikki Tauriainen},
  title   = {Automated Testing of {B\"u}chi Automata Translators for
    {L}inear {T}emporal {L}ogic},
  address = {Espoo, Finland},
  institution = {Helsinki University of Technology, Laboratory for
    Theoretical Computer Science},
  number = {A66},
  year = {2000},
  url = {http://citeseer.nj.nec.com/tauriainen00automated.html},
  type = {Research Report},
  note = {Reprint of Master's thesis}
}
```

Although the intent is similar, there are some differences with between the above published algorithm and this implementation . First labels are on transitions, and acceptance conditions are generated too. Second, the number of successors of a node is chosen in  $[1, n]$  following a normal distribution with mean  $1+(n-1)d$  and variance  $(n-1)d(1-d)$ . (This is less accurate, but faster than considering all possible  $n$  successors one by one.)

#### 8.23.1.2 `tgba_statistics stats_reachable (const tgba * g)`

Compute statistics for an automaton.

#### 8.23.1.3 `tgba_explicit* tgba_dupexp_bfs (const tgba * aut)`

Build an explicit automata from all states of *aut*, numbering states in bread first order as they are processed.

#### 8.23.1.4 `tgba_explicit* tgba_dupexp_dfs (const tgba * aut)`

Build an explicit automata from all states of *aut*, numbering states in depth first order as they are processed.

### 8.23.1.5 `tgba_explicit* tgba_powerset (const tgba * aut)`

Build a deterministic automaton, ignoring acceptance conditions.

This create a deterministic automaton that recognize the same language as *aut* would if its acceptance conditions were ignored. This is the classical powerset algorithm.

## 8.24 Decorating the dot output

### Classes

- class `spot::dotty_decorator`  
*Choose state and link styles for `spot::dotty_reachable`.*
- class `spot::tgba_run_dotty_decorator`  
*Highlight a `spot::tgba_run` on a `spot::tgba`.*

## 8.25 Emptiness-checks

### Modules

- [Emptiness-check algorithms for SSP](#)
- [Emptiness-check algorithms](#)
- [TGBA runs and supporting functions](#)
- [Emptiness-check statistics](#)

### Classes

- class `spot::emptiness_check_result`  
*The result of an emptiness check.*
- class `spot::emptiness_check`  
*Common interface to emptiness check algorithms.*

### 8.25.1 Detailed Description

All emptiness-check algorithms follow the same interface. Basically once you have constructed an instance of `spot::emptiness_check` (by instantiating a subclass, or calling a functions construct such instance; see [this list](#)), you should call `spot::emptiness_check::check()` to check the automaton.

If `spot::emptiness_check::check()` returns 0, then the automaton was found empty. Otherwise the automaton accepts some run. (Beware that some algorithms—those using bit-state hashing—may found the automaton to be empty even if it is not actually empty.)

When `spot::emptiness_check::check()` does not return 0, it returns an instance of `spot::emptiness_check_result`. You can try to call `spot::emptiness_check_result::accepting_run()` to obtain an accepting run. For some emptiness-check algorithms, `spot::emptiness_check_result::accepting_run()` will require some extra computation. Most emptiness-check algorithms are able to return such an accepting run, however this is not mandatory and `spot::emptiness_check_result::accepting_run()` can return 0 (this does not means by anyway that no accepting run exist).

The acceptance run returned by `spot::emptiness_check_result::accepting_run()`, if any, is of type `spot::tgba_run`. [This page](#) gathers existing operations on these objects.

## 8.26 Emptiness-check algorithms

### Classes

- class `spot::couvreur99_check`  
*Check whether the language of an automaton is empty.*
- class `spot::couvreur99_check_shy`  
*A version of `spot::couvreur99_check` that tries to visit known states first.*

### Functions

- `emptiness_check * spot::explicit_gv04_check (const tgba *a)`  
*Emptiness check based on Geldenhuys and Valmari's TACAS'04 paper.*
- `emptiness_check * spot::explicit_magic_search (const tgba *a)`  
*Returns an emptiness checker on the `spot::tgba` automaton `a`.*
- `emptiness_check * spot::bit_state_hashing_magic_search (const tgba *a, size_t size)`  
*Returns an emptiness checker on the `spot::tgba` automaton `a`.*
- `emptiness_check * spot::explicit_se05_search (const tgba *a)`  
*Returns an emptiness check on the `spot::tgba` automaton `a`.*
- `emptiness_check * spot::bit_state_hashing_se05_search (const tgba *a, size_t size)`  
*Returns an emptiness checker on the `spot::tgba` automaton `a`.*
- `emptiness_check * spot::explicit_tau03_search (const tgba *a)`  
*Returns an emptiness checker on the `spot::tgba` automaton `a`.*
- `emptiness_check * spot::explicit_tau03_opt_search (const tgba *a)`  
*Returns an emptiness checker on the `spot::tgba` automaton `a`.*

### 8.26.1 Function Documentation

#### 8.26.1.1 `emptiness_check * bit_state_hashing_magic_search (const tgba *a, size_t size)`

Returns an emptiness checker on the `spot::tgba` automaton `a`.

#### Precondition:

The automaton `a` must have at most one accepting condition (i.e. it is a TBA).

During the visit of `a`, the returned checker does not store explicitly the traversed states but uses the bit-state hashing technic presented in:

```
@book{Holzmann91,
  author = {G.J. Holzmann},
  title = {Design and Validation of Computer Protocols},
  publisher = {Prentice-Hall},
  address = {Englewood Cliffs, New Jersey},
  year = {1991}
}
```

Consequently, the detection of an acceptance cycle is not ensured. The implemented algorithm is the same as the one of [spot::explicit\\_magic\\_search](#).

**See also:**

[spot::explicit\\_magic\\_search](#)

### 8.26.1.2 emptiness\_check\* bit\_state\_hashing\_se05\_search (const tgba \* *a*, size\_t *size*)

Returns an emptiness checker on the [spot::tgba](#) automaton *a*.

**Precondition:**

The automaton *a* must have at most one accepting condition (i.e. it is a TBA).

During the visit of *a*, the returned checker does not store explicitly the traversed states but uses the bit-state hashing technic presented in:

```
@book{Holzmann91,
  author = {G.J. Holzmann},
  title = {Design and Validation of Computer Protocols},
  publisher = {Prentice-Hall},
  address = {Englewood Cliffs, New Jersey},
  year = {1991}
}
```

Consequently, the detection of an acceptance cycle is not ensured. The implemented algorithm is the same as the one of [spot::explicit\\_se05\\_search](#).

**See also:**

[spot::explicit\\_se05\\_search](#)

### 8.26.1.3 emptiness\_check\* explicit\_gv04\_check (const tgba \* *a*)

Emptiness check based on Geldenhuys and Valmari's TACAS'04 paper.

**Precondition:**

The automaton *a* must have at most one accepting condition.

The original algorithm, coming from the following paper, has only been slightly modified to work on transition-based automata.

```
@InProceedings{geldenhuys.04.tacas,
  author = {Jaco Geldenhuys and Antti Valmari},
  title = {Tarjan's Algorithm Makes On-the-Fly {LTL} Verification
    More Efficient},
```

```

booktitle = {Proceedings of the 10th International Conference on Tools
             and Algorithms for the Construction and Analysis of Systems
             (TACAS'04)},
editor    = {Kurt Jensen and Andreas Podelski},
pages     = {205--219},
year      = {2004},
publisher = {Springer-Verlag},
series    = {Lecture Notes in Computer Science},
volume    = {2988},
isbn      = {3-540-21299-X}
}

```

#### 8.26.1.4 emptiness\_check\* explicit\_magic\_search (const tgba \* a)

Returns an emptiness checker on the `spot::tgba` automaton *a*.

##### Precondition:

The automaton *a* must have at most one accepting condition (i.e. it is a TBA).

During the visit of *a*, the returned checker stores explicitly all the traversed states. The method *check()* of the checker can be called several times (until it returns a null pointer) to enumerate all the visited accepting paths. The implemented algorithm is the following:

```

procedure check ()
begin
  call dfs_blue(s0);
end;

procedure dfs_blue (s)
begin
  s.color = blue;
  for all t in post(s) do
    if t.color == white then
      call dfs_blue(t);
    end if;
    if (the edge (s,t) is accepting) then
      target = s;
      call dfs_red(t);
    end if;
  end for;
end;

procedure dfs_red(s)
begin
  s.color = red;
  if s == target then
    report cycle
  end if;
  for all t in post(s) do
    if t.color == blue then
      call dfs_red(t);
    end if;
  end for;
end;

```

This algorithm is an adaptation to TBA of the one (which deals with accepting states) presented in

```

Article{
  author      = {Costas Courcoubetis and Moshe Y. Vardi and Pierre
                courcoubertis.92.fmsd,

```

```

        Wolper and Mihalīs Yannakakis},
title      = {Memory-Efficient Algorithm for the Verification of
              Temporal Properties},
journal    = {Formal Methods in System Design},
pages      = {275--288},
year       = {1992},
volume     = {1}
}

```

#### 8.26.1.5 emptiness\_check\* explicit\_se05\_search (const tgba \* a)

Returns an emptiness check on the [spot::tgba](#) automaton *a*.

##### Precondition:

The automaton *a* must have at most one accepting condition (i.e. it is a TBA).

During the visit of *a*, the returned checker stores explicitly all the traversed states. The method *check()* of the checker can be called several times (until it returns a null pointer) to enumerate all the visited accepting paths. The implemented algorithm is an optimization of [spot::explicit\\_magic\\_search](#) and is the following:

```

procedure check ()
begin
  call dfs_blue(s0);
end;

procedure dfs_blue (s)
begin
  s.color = cyan;
  for all t in post(s) do
    if t.color == white then
      call dfs_blue(t);
    else if t.color == cyan and
      (the edge (s,t) is accepting or
       (it exists a predecessor p of s in st_blue and s != t and
        the arc between p and s is accepting)) then
      report cycle;
    end if;
    if the edge (s,t) is accepting then
      call dfs_red(t);
    end if;
  end for;
  s.color = blue;
end;

procedure dfs_red(s)
begin
  if s.color == cyan then
    report cycle;
  end if;
  s.color = red;
  for all t in post(s) do
    if t.color == blue then
      call dfs_red(t);
    end if;
  end for;
end;

```

It is an adaptation to TBA of the one presented in

```

@techreport{SE04,
  author = {Stefan Schwoon and Javier Esparza},
  institution = {Universit{"a"}t Stuttgart, Fakult{"a"}t Informatik,
  Elektrotechnik und Informationstechnik},
  month = {November},
  number = {2004/06},
  title = {A Note on On-The-Fly Verification Algorithms},
  year = {2004},
  url =
{http://www.fmi.uni-stuttgart.de/szs/publications/info/schwoosn.SE04.shtml}
}

```

**See also:**

[spot::explicit\\_magic\\_search](#)

#### 8.26.1.6 emptiness\_check\* explicit\_tau03\_opt\_search (const tgba \* a)

Returns an emptiness checker on the [spot::tgba](#) automaton  $a$ .

**Precondition:**

The automaton  $a$  must have at least one accepting condition.

During the visit of  $a$ , the returned checker stores explicitly all the traversed states. The implemented algorithm is the following:

```

procedure check ()
begin
  weight = 0; // the null vector
  call dfs_blue(s0);
end;

procedure dfs_blue (s)
begin
  s.color = cyan;
  s.acc = emptyset;
  s.weight = weight;
  for all t in post(s) do
    let (s, l, a, t) be the edge from s to t;
    if t.color == white then
      for all b in a do
        weight[b] = weight[b] + 1;
      end for;
      call dfs_blue(t);
      for all b in a do
        weight[b] = weight[b] - 1;
      end for;
    end if;
    Acc = s.acc U a;
    if t.color == cyan &&
      (Acc U support(weight - t.weight) U t.acc) == all_acc then
      report a cycle;
    else if Acc not included in t.acc then
      t.acc := t.acc U Acc;
      call dfs_red(t, Acc);
    end if;
  end for;
  s.color = blue;
end;

procedure dfs_red(s, Acc)
begin

```

```

for all t in post(s) do
  let (s, l, a, t) be the edge from s to t;
  if t.color == cyan &&
    (Acc U support(weight - t.weight) U t.acc) == all_acc then
    report a cycle;
  else if t.color != white and Acc not included in t.acc then
    t.acc := t.acc U Acc;
    call dfs_red(t, Acc);
  end if;
end for;
end;

```

This algorithm is a generalisation to TGBA of the one implemented in [spot::explicit\\_se05\\_search](#). It is based on the acceptance set labelling of states used in [spot::explicit\\_tau03\\_search](#). Moreover, it introduces a slight optimisation based on vectors of integers counting for each acceptance condition how many times the condition has been visited in the path stored in the blue stack. Such a vector is associated to each state of this stack.

#### 8.26.1.7 emptiness\_check\* explicit\_tau03\_search (const tgba \* a)

Returns an emptiness checker on the [spot::tgba](#) automaton  $a$ .

##### Precondition:

The automaton  $a$  must have at least one accepting condition.

During the visit of  $a$ , the returned checker stores explicitly all the traversed states. The implemented algorithm is the following:

```

procedure check ()
begin
  call dfs_blue(s0);
end;

procedure dfs_blue (s)
begin
  s.color = blue;
  s.acc = emptyset;
  for all t in post(s) do
    if t.color == white then
      call dfs_blue(t);
    end if;
  end for;
  for all t in post(s) do
    let (s, l, a, t) be the edge from s to t;
    if s.acc U a not included in t.acc then
      call dfs_red(t, a U s.acc);
    end if;
  end for;
  if s.acc == all_acc then
    report a cycle;
  end if;
end;

procedure dfs_red(s, A)
begin
  s.acc = s.acc U A;
  for all t in post(s) do
    if t.color != white and A not included in t.acc then
      call dfs_red(t, A);
    end if;
  end for;
end;

```



```
end;
```

This algorithm is the one presented in

```
@techreport{HUT-TCS-A83,
  address = {Espoo, Finland},
  author = {Heikki Tauriainen},
  institution = {Helsinki University of Technology, Laboratory for
    Theoretical Computer Science},
  month = {December},
  number = {A83},
  pages = {132},
  title = {On Translating Linear Temporal Logic into Alternating and
    Nondeterministic Automata},
  type = {Research Report},
  year = {2003},
  url = {http://www.tcs.hut.fi/Publications/info/bibdb.HUT-TCS-A83.shtml}
}
```

## 8.27 TGBA runs and supporting functions

### Classes

- struct [spot::tgba\\_run](#)  
*An accepted run, for a tgba.*

### Functions

- `std::ostream & spot::print_tgba_run (std::ostream &os, const tgba *a, const tgba_run *run)`  
*Display a [tgba\\_run](#).*
- `tgba * spot::tgba_run_to_tgba (const tgba *a, const tgba_run *run)`  
*Return an explicit tgba corresponding to run (i.e. comparable states are merged).*
- `tgba_run * spot::project_tgba_run (const tgba *a_run, const tgba *a_proj, const tgba_run *run)`  
*Project a [tgba\\_run](#) on a tgba.*
- `tgba_run * spot::reduce_run (const tgba *a, const tgba_run *org)`  
*Reduce an accepting run.*
- `bool spot::replay_tgba_run (std::ostream &os, const tgba *a, const tgba_run *run, bool debug=false)`  
*Replay a [tgba\\_run](#) on a tgba.*

### 8.27.1 Function Documentation

#### 8.27.1.1 `std::ostream& print_tgba_run (std::ostream & os, const tgba * a, const tgba_run * run)`

Display a [tgba\\_run](#).

Output the prefix and cycle of the `tgba_run` *run*, even if it does not corresponds to an actual run of the automaton *a*. This is unlike `replay_tgba_run()`, which will ensure the run actually exist in the automaton (and will display any transition annotation).

(*a* is used here only to format states and transitions.)

Output the prefix and cycle of the `tgba_run` *run*, even if it does not corresponds to an actual run of the automaton *a*. This is unlike `replay_tgba_run()`, which will ensure the run actually exist in the automaton (and will display any transition annotation).

#### 8.27.1.2 `tgba_run* project_tgba_run (const tgba * a_run, const tgba * a_proj, const tgba_run * run)`

Project a `tgba_run` on a `tgba`.

If a `tgba_run` has been generated on a product, or any other on-the-fly algorithm with `tgba` operands,

##### Parameters:

- run* the run to replay
- a\_run* the automata on which the run was generated
- a\_proj* the automata on which to project the run

##### Returns:

true iff the run could be completed

#### 8.27.1.3 `tgba_run* reduce_run (const tgba * a, const tgba_run * org)`

Reduce an accepting run.

Return a run which is accepting for *and* that is no longer that *org*.

#### 8.27.1.4 `bool replay_tgba_run (std::ostream & os, const tgba * a, const tgba_run * run, bool debug = false)`

Replay a `tgba_run` on a `tgba`.

This is similar to `print_tgba_run()`, except that the run is actually replayed on the automaton while it is printed. Doing so makes it possible to display transition annotations (returned by `spot::tgba::transition_annotation()`). The output will stop if the run cannot be completed.

##### Parameters:

- run* the run to replay
- a* the automata on which to replay that run
- os* the stream on which the replay should be traced
- debug* if set the output will be more verbose and extra debugging informations will be output on failure

##### Returns:

true iff the run could be completed

#### 8.27.1.5 `tgba* tgba_run_to_tgba (const tgba * a, const tgba_run * run)`

Return an explicit `tgba` corresponding to *run* (i.e. comparable states are merged).

##### Precondition:

*run* must correspond to an actual run of the automaton *a*.

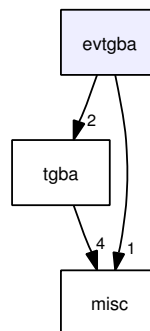
## 8.28 Emptiness-check statistics

### Classes

- class [spot::ec\\_statistics](#)  
*Emptiness-check statistics.*
- class [spot::acss\\_statistics](#)  
*Accepting Cycle Search Space statistics.*
- class [spot::ars\\_statistics](#)  
*Accepting Run Search statistics.*

## 9 spot Directory Documentation

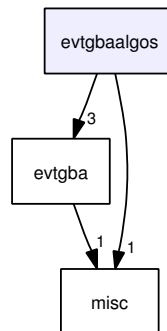
### 9.1 evtgba/ Directory Reference



### Files

- file [evtgba.hh](#)
- file [evtgbaiter.hh](#)
- file [explicit.hh](#)
- file [product.hh](#)
- file [symbol.hh](#)

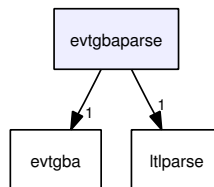
## 9.2 evtgbaalgos/ Directory Reference



### Files

- file [dotty.hh](#)
- file [reachiter.hh](#)
- file [save.hh](#)
- file [tgba2evtgba.hh](#)

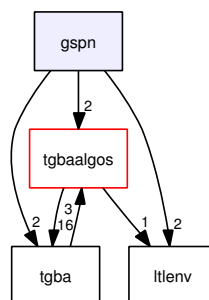
## 9.3 evtgbaparse/ Directory Reference



### Files

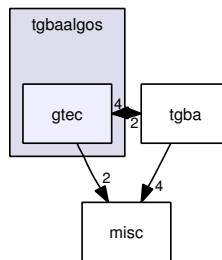
- file [public.hh](#)

## 9.4 gspn/ Directory Reference

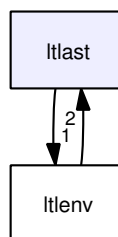


**Files**

- file [common.hh](#)
- file [gspn.hh](#)
- file [ssp.hh](#)

**9.5 tgbaalgos/gtec/ Directory Reference****Files**

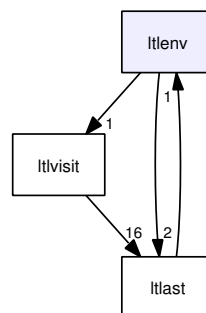
- file [ce.hh](#)
- file [explscc.hh](#)
- file [gtec.hh](#)
- file [nsheap.hh](#)
- file [sccstack.hh](#)
- file [status.hh](#)

**9.6 Itlast/ Directory Reference****Files**

- file [allnodes.hh](#)  
*Define all LTL node types.*
- file [atomic\\_prop.hh](#)  
*LTL atomic propositions.*

- file [binop.hh](#)  
*LTL binary operators.*
- file [constant.hh](#)  
*LTL constants.*
- file [formula.hh](#)  
*LTL formula interface.*
- file [multop.hh](#)  
*LTL multi-operand operators.*
- file [predecl.hh](#)  
*Predeclare all LTL node types.*
- file [reformula.hh](#)  
*Reference-counted LTL formulae.*
- file [unop.hh](#)  
*LTL unary operators.*
- file [visitor.hh](#)  
*LTL visitor interface.*

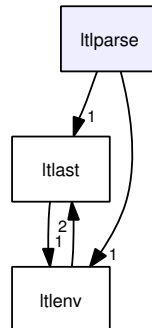
## 9.7 Itlenv/ Directory Reference



### Files

- file [declenv.hh](#)
- file [defaultenv.hh](#)
- file [environment.hh](#)

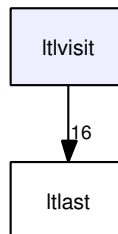
## 9.8 Itlparse/ Directory Reference



### Files

- file [location.hh](#)
- file [position.hh](#)
- file [public.hh](#)
- file [stack.hh](#)

## 9.9 Itlvisit/ Directory Reference



### Files

- file [apcollect.hh](#)
- file [basicreduce.hh](#)
- file [clone.hh](#)
- file [destroy.hh](#)
- file [dotty.hh](#)
- file [dump.hh](#)
- file [length.hh](#)
- file [lunabbrev.hh](#)
- file [neniform.hh](#)
- file [postfix.hh](#)
- file [randoml.tl.hh](#)
- file [reduce.hh](#)
- file [simpfg.hh](#)

- file [syntimpl.hh](#)
- file [tostring.hh](#)
- file [tunabbrev.hh](#)

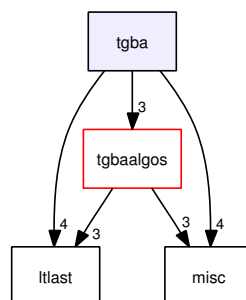
## 9.10 misc/ Directory Reference



### Files

- file [bareword.hh](#)
- file [bddalloc.hh](#)
- file [bddlt.hh](#)
- file [escape.hh](#)
- file [freelist.hh](#)
- file [hash.hh](#)
- file [hashfunc.hh](#)
- file [minato.hh](#)
- file [modgray.hh](#)
- file [random.hh](#)
- file [timer.hh](#)
- file [version.hh](#)

## 9.11 tgba/ Directory Reference



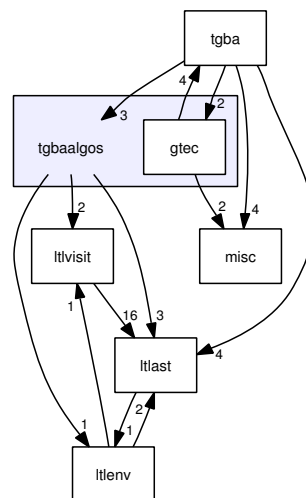
### Files

- file [bdddict.hh](#)
- file [bddprint.hh](#)
- file [formula2bdd.hh](#)
- file [public.hh](#)
- file [state.hh](#)
- file [statebdd.hh](#)



- file [succiter.hh](#)
- file [succiterconcrete.hh](#)
- file [tgba.hh](#)
- file [tgbabddconcrete.hh](#)
- file [tgbabddconcretefactory.hh](#)
- file [tgbabddconcreteproduct.hh](#)
- file [tgbabddcoredata.hh](#)
- file [tgbabddfactory.hh](#)
- file [tgbaexplicit.hh](#)
- file [tgbaproduct.hh](#)
- file [tgbaeduc.hh](#)
- file [tgbatba.hh](#)

## 9.12 tgbaalgos/ Directory Reference



### Directories

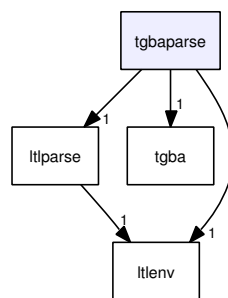
- directory [gtec](#)

### Files

- file [bfssteps.hh](#)
- file [dotty.hh](#)
- file [dottydec.hh](#)
- file [dupexp.hh](#)
- file [emptiness.hh](#)
- file [emptiness\\_stats.hh](#)
- file [gv04.hh](#)
- file [lbtt.hh](#)
- file [ltl2tgba\\_fm.hh](#)
- file [ltl2tgba\\_lacim.hh](#)

- file [magic.hh](#)
- file [neverclaim.hh](#)
- file [powerset.hh](#)
- file [projrun.hh](#)
- file [randomgraph.hh](#)
- file [reachiter.hh](#)
- file [reducerun.hh](#)
- file [reductgba\\_sim.hh](#)
- file [replayrun.hh](#)
- file [rundotdec.hh](#)
- file [save.hh](#)
- file [se05.hh](#)
- file [stats.hh](#)
- file [tau03.hh](#)
- file [tau03opt.hh](#)
- file [weight.hh](#)

### 9.13 tgbaparse/ Directory Reference



#### Files

- file [public.hh](#)

## 10 spot Namespace Documentation

### 10.1 spot Namespace Reference

#### Classes

- class [evtgba](#)
- class [evtgba\\_iterator](#)
- class [evtgba\\_explicit](#)
- class [state\\_evtgba\\_explicit](#)  
*States used by spot::tgba::evtgba\_explicit.*
- class [evtgba\\_product](#)
- class [symbol](#)

- class [rsymbol](#)
- class [evtgba\\_reachable\\_iterator](#)  
*Iterate over all reachable states of a [spot::evtgba](#).*
- class [evtgba\\_reachable\\_iterator\\_depth\\_first](#)  
*An implementation of [spot::evtgba\\_reachable\\_iterator](#) that browses states depth first.*
- class [evtgba\\_reachable\\_iterator\\_breadth\\_first](#)  
*An implementation of [spot::evtgba\\_reachable\\_iterator](#) that browses states breadth first.*
- class [bdd\\_allocator](#)  
*Manage ranges of variables.*
- struct [bdd\\_less\\_than](#)  
*Comparison functor for BDDs.*
- class [free\\_list](#)  
*Manage list of free integers.*
- struct [ptr\\_hash](#)  
*A hash function for pointers.*
- struct [string\\_hash](#)  
*A hash function for strings.*
- class [minato\\_isop](#)  
*Generate an irredundant sum-of-products (ISOP) form of a BDD function.*
- class [loopless\\_modular\\_mixed\\_radix\\_gray\\_code](#)  
*Loopless modular mixed radix Gray code iteration.*
- class [barand](#)  
*Compute pseudo-random integer value between 0 and n included, following a binomial distribution for probability p.*
- struct [time\\_info](#)  
*A structure to record elapsed time in clock ticks.*
- class [timer](#)  
*A timekeeper that accumulate interval of time.*
- class [timer\\_map](#)  
*A map of timer, where each timer has a name.*
- class [bdd\\_dict](#)
- class [state](#)  
*Abstract class for states.*
- struct [state\\_ptr\\_less\\_than](#)  
*Strict Weak Ordering for state\*.*

- struct [state\\_ptr\\_equal](#)  
*An Equivalence Relation for state\*.*
- struct [state\\_ptr\\_hash](#)  
*Hash Function for state\*.*
- class [state\\_bdd](#)
- class [tgba\\_succ\\_iterator](#)  
*Iterate over the successors of a state.*
- class [tgba\\_succ\\_iterator\\_concrete](#)
- class [tgba](#)  
*A Transition-based Generalized Büchi Automaton.*
- class [tgba\\_bdd\\_concrete](#)  
*A concrete [spot::tgba](#) implemented using BDDs.*
- class [tgba\\_bdd\\_concrete\\_factory](#)  
*Helper class to build a [spot::tgba\\_bdd\\_concrete](#) object.*
- struct [tgba\\_bdd\\_core\\_data](#)  
*Core data for a TGBA encoded using BDDs.*
- class [tgba\\_bdd\\_factory](#)  
*Abstract class for [spot::tgba\\_bdd\\_concrete](#) factories.*
- class [tgba\\_explicit](#)
- class [state\\_explicit](#)
- class [tgba\\_explicit\\_succ\\_iterator](#)
- class [state\\_product](#)  
*A state for [spot::tgba\\_product](#).*
- class [tgba\\_succ\\_iterator\\_product](#)  
*Iterate over the successors of a product computed on the fly.*
- class [tgba\\_product](#)  
*A lazy product. (States are computed on the fly.).*
- class [direct\\_simulation\\_relation](#)
- class [delayed\\_simulation\\_relation](#)
- class [tgba\\_reduc](#)
- class [tgba\\_tba\\_proxy](#)  
*Degeneralize a [spot::tgba](#) on the fly, producing a TBA.*
- class [tgba\\_sba\\_proxy](#)  
*Degeneralize a [spot::tgba](#) on the fly, producing an SBA.*
- class [bfs\\_steps](#)  
*Make a BFS in a [spot::tgba](#) to compute a [tgba\\_run::steps](#).*

- class [dotty\\_decorator](#)  
*Choose state and link styles for [spot::dotty\\_reachable](#).*
- class [emptiness\\_check\\_result](#)  
*The result of an emptiness check.*
- class [emptiness\\_check](#)  
*Common interface to emptiness check algorithms.*
- struct [tgba\\_run](#)  
*An accepted run, for a tgba.*
- class [ec\\_statistics](#)  
*Emptiness-check statistics.*
- class [acss\\_statistics](#)  
*Accepting Cycle Search Space statistics.*
- class [ars\\_statistics](#)  
*Accepting Run Search statistics.*
- class [couvreur99\\_check\\_result](#)  
*Compute a counter example from a [spot::couvreur99\\_check\\_status](#).*
- class [explicit\\_connected\\_component](#)  
*An SCC storing all its states explicitly.*
- class [connected\\_component\\_hash\\_set](#)
- class [explicit\\_connected\\_component\\_factory](#)  
*Abstract factory for [explicit\\_connected\\_component](#).*
- class [connected\\_component\\_hash\\_set\\_factory](#)  
*Factory for [connected\\_component\\_hash\\_set](#).*
- class [couvreur99\\_check](#)  
*Check whether the language of an automate is empty.*
- class [couvreur99\\_check\\_shy](#)  
*A version of [spot::couvreur99\\_check](#) that tries to visit known states first.*
- class [numbered\\_state\\_heap\\_const\\_iterator](#)  
*Iterator on [numbered\\_state\\_heap](#) objects.*
- class [numbered\\_state\\_heap](#)  
*Keep track of a large quantity of indexed states.*
- class [numbered\\_state\\_heap\\_factory](#)  
*Abstract factory for [numbered\\_state\\_heap](#).*

- class [numbered\\_state\\_heap\\_hash\\_map](#)  
*A straightforward implementation of [numbered\\_state\\_heap](#) with a hash map.*
- class [numbered\\_state\\_heap\\_hash\\_map\\_factory](#)  
*Factory for [numbered\\_state\\_heap\\_hash\\_map](#).*
- class [scc\\_stack](#)
- class [couvreur99\\_check\\_status](#)  
*The status of the emptiness-check on success.*
- class [tgba\\_reachable\\_iterator](#)  
*Iterate over all reachable states of a [spot::tgba](#).*
- class [tgba\\_reachable\\_iterator\\_depth\\_first](#)  
*An implementation of [spot::tgba\\_reachable\\_iterator](#) that browses states depth first.*
- class [tgba\\_reachable\\_iterator\\_breadth\\_first](#)  
*An implementation of [spot::tgba\\_reachable\\_iterator](#) that browses states breadth first.*
- class [parity\\_game\\_graph](#)  
*Parity game graph which compute a simulation relation.*
- class [spoiler\\_node](#)  
*Spoiler node of parity game graph.*
- class [duplicator\\_node](#)  
*Duplicator node of parity game graph.*
- class [parity\\_game\\_graph\\_direct](#)  
*Parity game graph which compute the direct simulation relation.*
- class [spoiler\\_node\\_delayed](#)  
*Spoiler node of parity game graph for delayed simulation.*
- class [duplicator\\_node\\_delayed](#)  
*Duplicator node of parity game graph for delayed simulation.*
- class [parity\\_game\\_graph\\_delayed](#)
- class [tgba\\_run\\_dotty\\_decorator](#)  
*Highlight a [spot::tgba\\_run](#) on a [spot::tgba](#).*
- struct [tgba\\_statistics](#)
- class [weight](#)  
*Manage for a given automaton a vector of counter indexed by its acceptance condition.*
- class [gspn\\_exeption](#)  
*An exeption used to forward GSPN errors.*
- class [gspn\\_interface](#)
- class [gspn\\_ssp\\_interface](#)

## Namespaces

- namespace [ltl](#)

## Typedefs

- typedef std::set< const [symbol](#) \* > [symbol\\_set](#)
- typedef std::set< [rsymbol](#) > [rsymbol\\_set](#)
- typedef std::pair< [yy::Location](#), std::string > [evtgba\\_parse\\_error](#)  
*A parse diagnostic with its location.*
- typedef std::list< [evtgba\\_parse\\_error](#) > [evtgba\\_parse\\_error\\_list](#)  
*A list of parser diagnostics, as filled by parse.*
- typedef Sgi::pair< const [spot::state](#) \*, const [spot::state](#) \* > [state\\_couple](#)
- typedef Sgi::vector< [state\\_couple](#) \* > [simulation\\_relation](#)
- typedef Sgi::vector< [spoiler\\_node](#) \* > [sn\\_v](#)
- typedef Sgi::vector< [duplicator\\_node](#) \* > [dn\\_v](#)
- typedef Sgi::vector< const [state](#) \* > [s\\_v](#)
- typedef std::pair< [yy::Location](#), std::string > [tgba\\_parse\\_error](#)  
*A parse diagnostic with its location.*
- typedef std::list< [tgba\\_parse\\_error](#) > [tgba\\_parse\\_error\\_list](#)  
*A list of parser diagnostics, as filled by parse.*

## Enumerations

- enum [reduce\\_tgba\\_options](#) {  
[Reduce\\_None](#) = 0, [Reduce\\_quotient\\_Dir\\_Sim](#) = 1, [Reduce\\_transition\\_Dir\\_Sim](#) = 2, [Reduce\\_quotient\\_Del\\_Sim](#) = 4,  
[Reduce\\_transition\\_Del\\_Sim](#) = 8, [Reduce\\_Scc](#) = 16, [Reduce\\_All](#) = -1U }  
*Options for reduce.*

## Functions

- std::ostream & [dotty\\_reachable](#) (std::ostream &os, const [evtgba](#) \*g)  
*Print reachable states in dot format.*
- std::ostream & [evtgba\\_save\\_reachable](#) (std::ostream &os, const [evtgba](#) \*g)  
*Save reachable states in text format.*
- [evtgba\\_explicit](#) \* [tgba\\_to\\_evtgba](#) (const [tgba](#) \*a)  
*Convert a tgba into an evtgba.*
- [evtgba\\_explicit](#) \* [evtgba\\_parse](#) (const std::string &filename, [evtgba\\_parse\\_error\\_list](#) &error\_list, bool debug=false)  
*Build a [spot::evtgba\\_explicit](#) from a text file.*

- bool [format\\_evtgba\\_parse\\_errors](#) (std::ostream &os, [evtgba\\_parse\\_error\\_list](#) &error\_list)  
*Format diagnostics produced by [spot::evtgba\\_parse](#).*
- bool [is\\_bare\\_word](#) (const char \*str)
- std::string [quote\\_unless\\_bare\\_word](#) (const std::string &str)  
*Double-quote words that are not bare.*
- std::ostream & [escape\\_str](#) (std::ostream &os, const std::string &str)  
*Escape " and \ characters in str.*
- std::string [escape\\_str](#) (const std::string &str)  
*Escape " and \ characters in str.*
- size\_t [wang32\\_hash](#) (size\_t key)  
*Thomas Wang's 32 bit hash function.*
- void [srand](#) (unsigned int seed)  
*Reset the seed of the pseudo-random number generator.*
- int [rrand](#) (int min, int max)  
*Compute a pseudo-random integer value between min and max included.*
- int [mrand](#) (int max)  
*Compute a pseudo-random integer value between 0 and max-1 included.*
- double [drand](#) ()  
*Compute a pseudo-random double value between 0.0 and 1.0 (1.0 excluded).*
- double [nrand](#) ()  
*Compute a pseudo-random double value following a standard normal distribution. (Odeh & Evans).*
- double [bmrand](#) ()  
*Compute a pseudo-random double value following a standard normal distribution. (Box-Muller).*
- int [prand](#) (double p)  
*Return a pseudo-random positive integer value following a Poisson distribution with parameter p.*
- const char \* [version](#) ()  
*Return Spot's version.*
- std::ostream & [bdd\\_print\\_sat](#) (std::ostream &os, const [bdd\\_dict](#) \*dict, bdd b)  
*Print a BDD as a list of literals.*
- std::string [bdd\\_format\\_sat](#) (const [bdd\\_dict](#) \*dict, bdd b)  
*Format a BDD as a list of literals.*
- std::ostream & [bdd\\_print\\_acc](#) (std::ostream &os, const [bdd\\_dict](#) \*dict, bdd b)  
*Print a BDD as a list of acceptance conditions.*



- `std::ostream & bdd_print_accset` (`std::ostream &os`, `const bdd_dict *dict`, `bdd b`)  
*Print a BDD as a set of acceptance conditions.*
- `std::string bdd_format_accset` (`const bdd_dict *dict`, `bdd b`)  
*Format a BDD as a set of acceptance conditions.*
- `std::ostream & bdd_print_set` (`std::ostream &os`, `const bdd_dict *dict`, `bdd b`)  
*Print a BDD as a set.*
- `std::string bdd_format_set` (`const bdd_dict *dict`, `bdd b`)  
*Format a BDD as a set.*
- `std::ostream & bdd_print_formula` (`std::ostream &os`, `const bdd_dict *dict`, `bdd b`)  
*Print a BDD as a formula.*
- `std::string bdd_format_formula` (`const bdd_dict *dict`, `bdd b`)  
*Format a BDD as a formula.*
- `std::ostream & bdd_print_dot` (`std::ostream &os`, `const bdd_dict *dict`, `bdd b`)  
*Print a BDD as a diagram in doty format.*
- `std::ostream & bdd_print_table` (`std::ostream &os`, `const bdd_dict *dict`, `bdd b`)  
*Print a BDD as a table.*
- `bdd formula_to_bdd` (`const ltl::formula *f`, `bdd_dict *d`, `void *for_me`)
- `const ltl::formula * bdd_to_formula` (`bdd f`, `const bdd_dict *d`)
- `tgba_bdd_concrete * product` (`const tgba_bdd_concrete *left`, `const tgba_bdd_concrete *right`)  
*Multiplies two `spot::tgba_bdd_concrete` automata.*
- `std::ostream & dotty_reachable` (`std::ostream &os`, `const tgba *g`, `dotty_decorator *dd=dotty_decorator::instance()`)  
*Print reachable states in dot format.*
- `tgba_explicit * tgba_dupexp_bfs` (`const tgba *aut`)  
*Build an explicit automata from all states of aut, numbering states in bread first order as they are processed.*
- `tgba_explicit * tgba_dupexp_dfs` (`const tgba *aut`)  
*Build an explicit automata from all states of aut, numbering states in depth first order as they are processed.*
- `std::ostream & print_tgba_run` (`std::ostream &os`, `const tgba *a`, `const tgba_run *run`)  
*Display a `tgba_run`.*
- `tgba * tgba_run_to_tgba` (`const tgba *a`, `const tgba_run *run`)  
*Return an explicit\_tgba corresponding to run (i.e. comparable states are merged).*
- `emptiness_check * explicit_gv04_check` (`const tgba *a`)  
*Emptiness check based on Geldenhuys and Valmari's TACAS'04 paper.*
- `std::ostream & lbtt_reachable` (`std::ostream &os`, `const tgba *g`)  
*Print reachable states in LBTT format.*

- `tgba_explicit * ltl_to_tgba_fm` (const `ltl::formula` \*f, `bdd_dict` \*dict, bool `exprop`=false, bool `symb_merge`=true, bool `branching_postponement`=false, bool `fair_loop_approx`=false, const `ltl::atomic_prop_set` \*unobs=0)  
Build a `spot::tgba_explicit` from an LTL formula.
- `tgba_bdd_concrete * ltl_to_tgba_lacim` (const `ltl::formula` \*f, `bdd_dict` \*dict)  
Build a `spot::tgba_bdd_concrete` from an LTL formula.
- `emptiness_check * explicit_magic_search` (const `tgba` \*a)  
Returns an emptiness checker on the `spot::tgba` automaton a.
- `emptiness_check * bit_state_hashing_magic_search` (const `tgba` \*a, `size_t` size)  
Returns an emptiness checker on the `spot::tgba` automaton a.
- `std::ostream & never_claim_reachable` (std::ostream &os, const `tgba_sba_proxy` \*g, const `ltl::formula` \*f=0)  
Print reachable states in Spin never claim format.
- `tgba_explicit * tgba_powerset` (const `tgba` \*aut)  
Build a deterministic automaton, ignoring acceptance conditions.
- `tgba_run * project_tgba_run` (const `tgba` \*a\_run, const `tgba` \*a\_proj, const `tgba_run` \*run)  
Project a `tgba_run` on a `tgba`.
- `tgba * random_graph` (int n, float d, const `ltl::atomic_prop_set` \*ap, `bdd_dict` \*dict, int n\_acc=0, float a=0.1, float t=0.5, `ltl::environment` \*env=&ltl::default\_environment::instance())  
Construct a `tgba` randomly.
- `tgba_run * reduce_run` (const `tgba` \*a, const `tgba_run` \*org)  
Reduce an accepting run.
- `tgba * reduc_tgba_sim` (const `tgba` \*a, int opt=Reduce\_All)  
Remove some node of the automata using a simulation relation.
- `direct_simulation_relation * get_direct_relation_simulation` (const `tgba` \*a, std::ostream &os, int opt=-1)  
Compute a direct simulation relation on state of `tgba` f.
- `delayed_simulation_relation * get_delayed_relation_simulation` (const `tgba` \*a, std::ostream &os, int opt=-1)  
Compute a delayed simulation relation on state of `tgba` f.
- void `free_relation_simulation` (`direct_simulation_relation` \*rel)  
To free a simulation relation.
- void `free_relation_simulation` (`delayed_simulation_relation` \*rel)  
To free a simulation relation.
- bool `replay_tgba_run` (std::ostream &os, const `tgba` \*a, const `tgba_run` \*run, bool debug=false)

Replay a [tgba\\_run](#) on a [tgba](#).

- [std::ostream & tgba\\_save\\_reachable](#) (std::ostream &os, const [tgba](#) \*g)  
Save reachable states in text format.
- [emptiness\\_check \\* explicit\\_se05\\_search](#) (const [tgba](#) \*a)  
Returns an emptiness check on the [spot::tgba](#) automaton a.
- [emptiness\\_check \\* bit\\_state\\_hashing\\_se05\\_search](#) (const [tgba](#) \*a, size\_t size)  
Returns an emptiness checker on the [spot::tgba](#) automaton a.
- [tgba\\_statistics stats\\_reachable](#) (const [tgba](#) \*g)  
Compute statistics for an automaton.
- [emptiness\\_check \\* explicit\\_tau03\\_search](#) (const [tgba](#) \*a)  
Returns an emptiness checker on the [spot::tgba](#) automaton a.
- [emptiness\\_check \\* explicit\\_tau03\\_opt\\_search](#) (const [tgba](#) \*a)  
Returns an emptiness checker on the [spot::tgba](#) automaton a.
- [tgba\\_explicit \\* tgba\\_parse](#) (const std::string &filename, [tgba\\_parse\\_error\\_list](#) &error\_list, [bdd\\_dict](#) \*dict, [ltl::environment](#) &env=[ltl::default\\_environment::instance\(\)](#), bool debug=false)  
Build a [spot::tgba\\_explicit](#) from a text file.
- [bool format\\_tgba\\_parse\\_errors](#) (std::ostream &os, const std::string &filename, [tgba\\_parse\\_error\\_list](#) &error\_list)  
Format diagnostics produced by [spot::tgba\\_parse](#).
- [std::ostream & operator<<](#) (std::ostream &os, const [gspn\\_exception](#) &e)
- [couvereur99\\_check \\* couvreur99\\_check\\_ssp\\_semi](#) (const [tgba](#) \*ssp\_automata)
- [couvereur99\\_check \\* couvreur99\\_check\\_ssp\\_shy\\_semi](#) (const [tgba](#) \*ssp\_automata)
- [couvereur99\\_check \\* couvreur99\\_check\\_ssp\\_shy](#) (const [tgba](#) \*ssp\_automata)

### 10.1.1 Typedef Documentation

#### 10.1.1.1 typedef std::pair<[yy::Location](#), std::string> [spot::evtgba\\_parse\\_error](#)

A parse diagnostic with its location.

#### 10.1.1.2 typedef std::list<[evtgba\\_parse\\_error](#)> [spot::evtgba\\_parse\\_error\\_list](#)

A list of parser diagnostics, as filled by parse.

#### 10.1.1.3 typedef std::set<[rsymbol](#)> [spot::rsymbol\\_set](#)

#### 10.1.1.4 typedef Sgi::vector<[state\\_couple](#)> [spot::simulation\\_relation](#)

#### 10.1.1.5 typedef Sgi::pair<const [spot::state](#)\*, const [spot::state](#)> [spot::state\\_couple](#)

#### 10.1.1.6 typedef std::set<const [symbol\\*](#)> [spot::symbol\\_set](#)

### 10.1.2 Function Documentation

#### 10.1.2.1 std::string bdd\_format\_accset (const bdd\_dict \* *dict*, bdd *b*)

Format a BDD as a set of acceptance conditions.

This is used when saving a TGBA.

**Parameters:**

*dict* The dictionary to use, to lookup variables.

*b* The BDD to print.

**Returns:**

The BDD formatted as a string.

#### 10.1.2.2 std::string bdd\_format\_formula (const bdd\_dict \* *dict*, bdd *b*)

Format a BDD as a formula.

**Parameters:**

*dict* The dictionary to use, to lookup variables.

*b* The BDD to print.

**Returns:**

The BDD formatted as a string.

#### 10.1.2.3 std::string bdd\_format\_sat (const bdd\_dict \* *dict*, bdd *b*)

Format a BDD as a list of literals.

This assumes that *b* is a conjunction of literals.

**Parameters:**

*dict* The dictionary to use, to lookup variables.

*b* The BDD to print.

**Returns:**

The BDD formatted as a string.

#### 10.1.2.4 std::string bdd\_format\_set (const bdd\_dict \* *dict*, bdd *b*)

Format a BDD as a set.

**Parameters:**

*dict* The dictionary to use, to lookup variables.

*b* The BDD to print.

**Returns:**

The BDD formatted as a string.

**10.1.2.5 `std::ostream& bdd_print_acc (std::ostream & os, const bdd_dict * dict, bdd b)`**

Print a BDD as a list of acceptance conditions.

This is used when saving a TGBA.

**Parameters:**

*os* The output stream.

*dict* The dictionary to use, to lookup variables.

*b* The BDD to print.

**Returns:**

The BDD formatted as a string.

**10.1.2.6 `std::ostream& bdd_print_accset (std::ostream & os, const bdd_dict * dict, bdd b)`**

Print a BDD as a set of acceptance conditions.

This is used when saving a TGBA.

**Parameters:**

*os* The output stream.

*dict* The dictionary to use, to lookup variables.

*b* The BDD to print.

**Returns:**

The BDD formatted as a string.

**10.1.2.7 `std::ostream& bdd_print_dot (std::ostream & os, const bdd_dict * dict, bdd b)`**

Print a BDD as a diagram in dotty format.

**Parameters:**

*os* The output stream.

*dict* The dictionary to use, to lookup variables.

*b* The BDD to print.

**10.1.2.8 `std::ostream& bdd_print_formula (std::ostream & os, const bdd_dict * dict, bdd b)`**

Print a BDD as a formula.

**Parameters:**

*os* The output stream.

*dict* The dictionary to use, to lookup variables.

*b* The BDD to print.

**10.1.2.9 std::ostream& bdd\_print\_sat (std::ostream & *os*, const bdd\_dict \* *dict*, bdd *b*)**

Print a BDD as a list of literals.

This assumes that *b* is a conjunction of literals.

**Parameters:**

*os* The output stream.

*dict* The dictionary to use, to lookup variables.

*b* The BDD to print.

**10.1.2.10 std::ostream& bdd\_print\_set (std::ostream & *os*, const bdd\_dict \* *dict*, bdd *b*)**

Print a BDD as a set.

**Parameters:**

*os* The output stream.

*dict* The dictionary to use, to lookup variables.

*b* The BDD to print.

**10.1.2.11 std::ostream& bdd\_print\_table (std::ostream & *os*, const bdd\_dict \* *dict*, bdd *b*)**

Print a BDD as a table.

**Parameters:**

*os* The output stream.

*dict* The dictionary to use, to lookup variables.

*b* The BDD to print.

**10.1.2.12 const [ltl::formula](#)\* bdd\_to\_formula (bdd *f*, const bdd\_dict \* *d*)****10.1.2.13 std::ostream& dotty\_reachable (std::ostream & *os*, const evtgba \* *g*)**

Print reachable states in dot format.

**10.1.2.14 [evtgba\\_explicit](#)\* evtgba\_parse (const std::string & *filename*, [evtgba\\_parse\\_error\\_list](#) & *error\_list*, bool *debug* = false)**

Build a [spot::evtgba\\_explicit](#) from a text file.

**Parameters:**

*filename* The name of the file to parse.

*error\_list* A list that will be filled with parse errors that occurred during parsing.

*debug* When true, causes the parser to trace its execution.

**Returns:**

A pointer to the evtgba built from *filename*, or 0 if the file could not be opened.

Note that the parser usually tries to recover from errors. It can return a non zero value even if it encountered an error during the parsing of *filename*. If you want to make sure *filename* was parsed successfully, check *error\_list* for emptiness.

**Warning:**

This function is not reentrant.

**10.1.2.15** `std::ostream& evtgba_save_reachable (std::ostream & os, const evtgba * g)`

Save reachable states in text format.

**10.1.2.16** `bool format_evtgba_parse_errors (std::ostream & os, evtgba\_parse\_error\_list & error_list)`

Format diagnostics produced by [spot::evtgba\\_parse](#).

**Parameters:**

*os* Where diagnostics should be output.

*error\_list* The error list filled by [spot::ltl::parse](#) while parsing *ltl\_string*.

**Returns:**

`true` iff any diagnostic was output.

**10.1.2.17** `bdd formula_to_bdd (const ltl::formula * f, bdd_dict * d, void * for_me)`**10.1.2.18** `std::ostream& operator<< (std::ostream & os, const gspn_exception & e)`**10.1.2.19** `evtgba\_explicit* tgba_to_evtgba (const tgba * a)`

Convert a tgba into an evtgba.

(This cannot be done on-the-fly because the alphabet of a tgba is unknown beforehand.)

**10.2 spot::ltl Namespace Reference****Classes**

- class [atomic\\_prop](#)  
*Atomic propositions.*
- class [binop](#)  
*Binary operator.*
- class [constant](#)  
*A constant (True or False).*
- class [formula](#)  
*An LTL formula.*

- struct [formula\\_ptr\\_less\\_than](#)  
*Strict Weak Ordering for const formula\*.*
- struct [formula\\_ptr\\_hash](#)  
*Hash Function for const formula\*.*
- class [multop](#)  
*Multi-operand operators.*
- class [ref\\_formula](#)  
*A reference-counted LTL formula.*
- class [unop](#)  
*Unary operators.*
- struct [visitor](#)  
*Formula visitor that can modify the formula.*
- struct [const\\_visitor](#)  
*Formula visitor that cannot modify the formula.*
- class [declarative\\_environment](#)  
*A declarative environment.*
- class [default\\_environment](#)  
*A laxist environment.*
- class [environment](#)  
*An environment that describes atomic propositions.*
- class [clone\\_visitor](#)  
*Clone a formula.*
- class [unabbreviate\\_logic\\_visitor](#)  
*Clone and rewrite a formula to remove most of the abbreviated logical operators.*
- class [postfix\\_visitor](#)  
*Apply an algorithm on each node of an AST, during a postfix traversal.*
- class [random\\_ltl](#)  
*Generate random LTL formulae.*
- class [simplify\\_f\\_g\\_visitor](#)  
*Replace true U f and false R g by F f and G g.*
- class [unabbreviate\\_ltl\\_visitor](#)  
*Clone and rewrite a formula to remove most of the abbreviated LTL and logical operators.*



## Typedefs

- typedef std::pair< [yy::Location](#), std::string > [parse\\_error](#)  
*A parse diagnostic with its location.*
- typedef std::list< [parse\\_error](#) > [parse\\_error\\_list](#)  
*A list of parser diagnostics, as filled by parse.*
- typedef std::set< [atomic\\_prop](#) \*, [formula\\_ptr\\_less\\_than](#) > [atomic\\_prop\\_set](#)  
*Set of atomic propositions.*

## Enumerations

- enum [reduce\\_options](#) {  
    [Reduce\\_None](#) = 0, [Reduce\\_Basics](#) = 1, [Reduce\\_Syntactic\\_Implications](#) = 2, [Reduce\\_Eventuality\\_And\\_Universality](#) = 4,  
    [Reduce\\_All](#) = -1U }  
*Options for [spot::ltl::reduce](#).*

## Functions

- [formula](#) \* [parse](#) (const std::string &ltl\_string, [parse\\_error\\_list](#) &error\_list, [environment](#) &env=default\_environment::instance(), bool debug=false)  
*Build a formula from an LTL string.*
- bool [format\\_parse\\_errors](#) (std::ostream &os, const std::string &ltl\_string, [parse\\_error\\_list](#) &error\_list)  
*Format diagnostics produced by [spot::ltl::parse](#).*
- [atomic\\_prop\\_set](#) \* [atomic\\_prop\\_collect](#) (const [formula](#) \*f, [atomic\\_prop\\_set](#) \*s=0)  
*Return the set of atomic propositions occurring in a formula.*
- [formula](#) \* [basic\\_reduce](#) (const [formula](#) \*f)  
*Basic rewritings.*
- bool [is\\_GF](#) (const [formula](#) \*f)  
*Whether a formula starts with GF.*
- bool [is\\_FG](#) (const [formula](#) \*f)  
*Whether a formula starts with FG.*
- [formula](#) \* [clone](#) (const [formula](#) \*f)  
*Clone a formula.*
- void [destroy](#) (const [formula](#) \*f)  
*Destroys a formula.*

- `std::ostream & dotty (std::ostream &os, const formula *f)`  
*Write a formula tree using dot's syntax.*
- `std::ostream & dump (std::ostream &os, const formula *f)`  
*Dump a formula tree.*
- `int length (const formula *f)`  
*Compute the length of a formula.*
- `formula * unabbreviate\_logic (const formula *f)`  
*Clone and rewrite a formula to remove most of the abbreviated logical operators.*
- `formula * negative\_normal\_form (const formula *f, bool negated=false)`  
*Build the negative normal form of f.*
- `formula * reduce (const formula *f, int opt=Reduce_All)`  
*Reduce a formula f.*
- `bool is\_eventual (const formula *f)`  
*Check whether a formula is a pure eventuality.*
- `bool is\_universal (const formula *f)`  
*Check whether a formula is purely universal.*
- `formula * simplify\_f\_g (const formula *f)`  
*Replace `true ∪ f` and `false ∩ g` by `F f` and `G g`.*
- `bool syntactic\_implication (const formula *f1, const formula *f2)`  
*Syntactic implication.*
- `bool syntactic\_implication\_neg (const formula *f1, const formula *f2, bool right)`  
*Syntactic implication.*
- `std::ostream & to\_string (const formula *f, std::ostream &os)`  
*Output a formula as a (parsable) string.*
- `std::string to\_string (const formula *f)`  
*Convert a formula into a (parsable) string.*
- `std::ostream & to\_spin\_string (const formula *f, std::ostream &os)`  
*Output a formula as a (parsable by Spin) string.*
- `std::string to\_spin\_string (const formula *f)`  
*Convert a formula into a (parsable by Spin) string.*
- `formula * unabbreviate\_ltl (const formula *f)`  
*Clone and rewrite a formula to remove most of the abbreviated LTL and logical operators.*

### 10.2.1 Function Documentation

#### 10.2.1.1 `formula* unabbreviate_ltl (const formula *f)`

Clone and rewrite a formula to remove most of the abbreviated LTL and logical operators.

The rewriting performed on logical operator is the same as the one done by `spot::ltl::unabbreviate_logic`.

This will also rewrite unary operators such as `unop::F`, and `unop::G`, using only `binop::U`, and `binop::R`.

## 10.3 yy Namespace Reference

### Classes

- class `Location`  
*Abstract a `Location`.*
- class `Position`  
*Abstract a `Position`.*
- class `Stack`
- class `Slice`

### Functions

- `const Location operator+ (const Location &begin, const Location &end)`  
*Join two `Location` objects to create a `Location`.*
- `const Location operator+ (const Location &begin, unsigned int width)`  
*Add two `Location` objects.*
- `Location & operator+= (Location &res, unsigned int width)`  
*Add and assign a `Location`.*
- `std::ostream & operator<< (std::ostream &ostr, const Location &loc)`  
*Intercept output stream redirection.*
- `const Position & operator+= (Position &res, const int width)`  
*Add and assign a `Position`.*
- `const Position operator+ (const Position &begin, const int width)`  
*Add two `Position` objects.*
- `const Position & operator-= (Position &res, const int width)`  
*Add and assign a `Position`.*
- `const Position operator- (const Position &begin, const int width)`  
*Add two `Position` objects.*
- `std::ostream & operator<< (std::ostream &ostr, const Position &pos)`  
*Intercept output stream redirection.*

### 10.3.1 Function Documentation

#### 10.3.1.1 `const Position operator+ (const Position & begin, const int width)` [inline]

Add two [Position](#) objects.

#### 10.3.1.2 `const Location operator+ (const Location & begin, unsigned int width)` [inline]

Add two [Location](#) objects.

#### 10.3.1.3 `const Location operator+ (const Location & begin, const Location & end)` [inline]

Join two [Location](#) objects to create a [Location](#).

#### 10.3.1.4 `const Position& operator+= (Position & res, const int width)` [inline]

Add and assign a [Position](#).

#### 10.3.1.5 `Location& operator+= (Location & res, unsigned int width)` [inline]

Add and assign a [Location](#).

#### 10.3.1.6 `const Position operator- (const Position & begin, const int width)` [inline]

Add two [Position](#) objects.

#### 10.3.1.7 `const Position& operator-= (Position & res, const int width)` [inline]

Add and assign a [Position](#).

#### 10.3.1.8 `std::ostream& operator<< (std::ostream & ostr, const Position & pos)` [inline]

Intercept output stream redirection.

##### Parameters:

*ostr* the destination output stream

*pos* a reference to the [Position](#) to redirect

#### 10.3.1.9 `std::ostream& operator<< (std::ostream & ostr, const Location & loc)` [inline]

Intercept output stream redirection.

##### Parameters:

*ostr* the destination output stream

*loc* a reference to the [Location](#) to redirect

Avoid duplicate information.

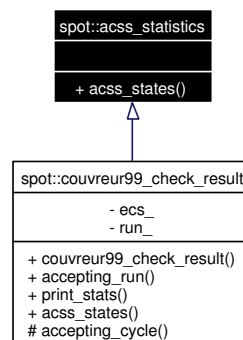
## 11 spot Class Documentation

### 11.1 spot::acss\_statistics Class Reference

Accepting Cycle Search Space statistics.

```
#include <tgbaalgos/emptiness_stats.hh>
```

Inheritance diagram for spot::acss\_statistics:



#### Public Member Functions

- virtual int [acss\\_states](#) () const =0  
*Number of states in the search space for the accepting cycle.*

#### 11.1.1 Detailed Description

Accepting Cycle Search Space statistics.

Implementations of [spot::emptiness\\_check\\_result](#) may also implement this interface. Try to `dynamic_cast` the [spot::emptiness\\_check\\_result](#) pointer to know whether these statistics are available.

#### 11.1.2 Member Function Documentation

##### 11.1.2.1 virtual int spot::acss\_statistics::acss\_states () const [pure virtual]

Number of states in the search space for the accepting cycle.

Implemented in [spot::couvreur99\\_check\\_result](#).

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

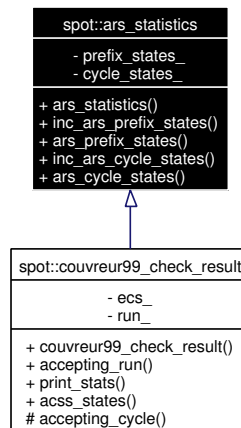
- [tgbaalgos/emptiness\\_stats.hh](#)

### 11.2 spot::ars\_statistics Class Reference

Accepting Run Search statistics.

```
#include <tgbaalgos/emptiness_stats.hh>
```

Inheritance diagram for spot::ars\_statistics:



### Public Member Functions

- [ars\\_statistics \(\)](#)
- void [inc\\_ars\\_prefix\\_states \(\)](#)
- int [ars\\_prefix\\_states \(\)](#) const
- void [inc\\_ars\\_cycle\\_states \(\)](#)
- int [ars\\_cycle\\_states \(\)](#) const

### Private Attributes

- unsigned [prefix\\_states\\_](#)
- unsigned [cycle\\_states\\_](#)  
*states visited to construct the prefix*

#### 11.2.1 Detailed Description

Accepting Run Search statistics.

Implementations of [spot::emptiness\\_check\\_result](#) may also implement this interface. Try to dynamic\_cast the [spot::emptiness\\_check\\_result](#) pointer to know whether these statistics are available.

#### 11.2.2 Constructor & Destructor Documentation

##### 11.2.2.1 spot::ars\_statistics::ars\_statistics () [inline]

#### 11.2.3 Member Function Documentation

##### 11.2.3.1 int spot::ars\_statistics::ars\_cycle\_states () const [inline]

##### 11.2.3.2 int spot::ars\_statistics::ars\_prefix\_states () const [inline]

11.2.3.3 void spot::ars\_statistics::inc\_ars\_cycle\_states () [inline]

11.2.3.4 void spot::ars\_statistics::inc\_ars\_prefix\_states () [inline]

#### 11.2.4 Member Data Documentation

11.2.4.1 unsigned [spot::ars\\_statistics::cycle\\_states\\_](#) [private]

states visited to construct the prefix

11.2.4.2 unsigned [spot::ars\\_statistics::prefix\\_states\\_](#) [private]

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

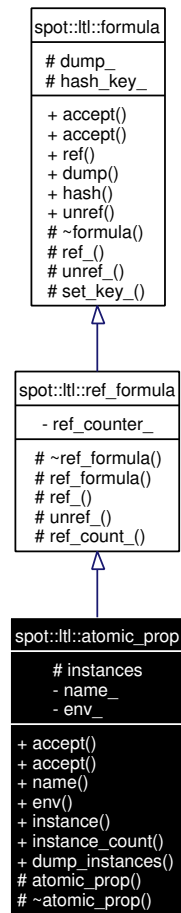
- [tgbaalgorithms/emptiness\\_stats.hh](#)

### 11.3 spot::ltl::atomic\_prop Class Reference

Atomic propositions.

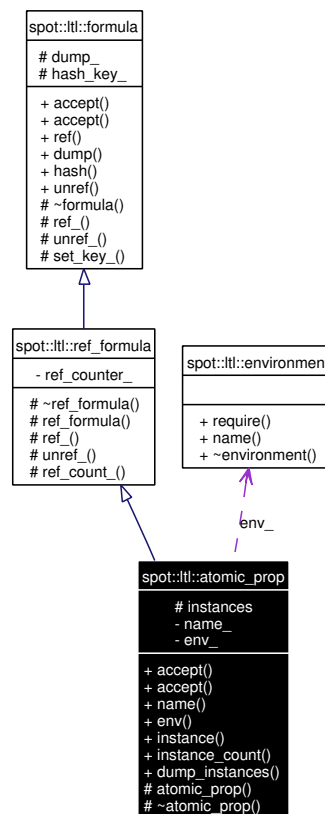
```
#include <ltlast/atomic_prop.hh>
```

Inheritance diagram for spot::ltl::atomic\_prop:



Collaboration diagram for `spot::ltl::atomic_prop`:





## Public Member Functions

- virtual void `accept (visitor &visitor)`  
Entry point for `vspot::ltl::visitor` instances.
- virtual void `accept (const_visitor &visitor) const`  
Entry point for `vspot::ltl::const_visitor` instances.
- const std::string & `name ()` const  
Get the name of the atomic proposition.
- `environment` & `env ()` const  
Get the environment of the atomic proposition.
- `formula * ref ()`  
clone this node
- const std::string & `dump ()` const  
Return a canonic representation of the formula.
- const size\_t `hash ()` const  
Return a `hash_key` for the formula.

### Static Public Member Functions

- static [atomic\\_prop](#) \* [instance](#) (const std::string &name, [environment](#) &env)
- static unsigned [instance\\_count](#) ()  
*Number of instantiated atomic propositions. For debugging.*
- static std::ostream & [dump\\_instances](#) (std::ostream &os)  
*List all instances of atomic propositions. For debugging.*
- static void [unref](#) ([formula](#) \*f)  
*release this node*

### Protected Types

- typedef std::pair< std::string, [environment](#) \* > [pair](#)
- typedef std::map< [pair](#), [atomic\\_prop](#) \* > [map](#)

### Protected Member Functions

- [atomic\\_prop](#) (const std::string &name, [environment](#) &env)
- virtual [~atomic\\_prop](#) ()
- void [ref\\_](#) ()  
*increment reference counter if any*
- bool [unref\\_](#) ()  
*decrement reference counter if any, return true when the instance must be deleted (usually when the counter hits 0).*
- unsigned [ref\\_count\\_](#) ()  
*Number of references to this formula.*
- void [set\\_key\\_](#) ()  
*Compute key\_ from dump\_.*

### Protected Attributes

- std::string [dump\\_](#)  
*The canonic representation of the formula.*
- size\_t [hash\\_key\\_](#)  
*The hash key of this formula.*

### Static Protected Attributes

- static [map](#) [instances](#)

**Private Attributes**

- std::string [name\\_](#)
- [environment](#) \* [env\\_](#)

**11.3.1 Detailed Description**

Atomic propositions.

**11.3.2 Member Typedef Documentation**

**11.3.2.1** typedef std::map<[pair](#), [atomic\\_prop](#)\*> [spot::ltl::atomic\\_prop::map](#) [protected]

**11.3.2.2** typedef std::pair<std::string, [environment](#)\*> [spot::ltl::atomic\\_prop::pair](#) [protected]

**11.3.3 Constructor & Destructor Documentation**

**11.3.3.1** [spot::ltl::atomic\\_prop::atomic\\_prop](#) (const std::string & *name*, [environment](#) & *env*) [protected]

**11.3.3.2** virtual [spot::ltl::atomic\\_prop::~~atomic\\_prop](#) () [protected, virtual]

**11.3.4 Member Function Documentation**

**11.3.4.1** virtual void [spot::ltl::atomic\\_prop::accept](#) ([const\\_visitor](#) & *visitor*) const [virtual]

Entry point for [vspot::ltl::const\\_visitor](#) instances.

Implements [spot::ltl::formula](#).

**11.3.4.2** virtual void [spot::ltl::atomic\\_prop::accept](#) ([visitor](#) & *visitor*) [virtual]

Entry point for [vspot::ltl::visitor](#) instances.

Implements [spot::ltl::formula](#).

**11.3.4.3** const std::string& [spot::ltl::formula::dump](#) () const [inherited]

Return a canonic representation of the formula.

**11.3.4.4** static std::ostream& [spot::ltl::atomic\\_prop::dump\\_instances](#) (std::ostream & *os*) [static]

List all instances of atomic propositions. For debugging.

**11.3.4.5** [environment](#)& [spot::ltl::atomic\\_prop::env](#) () const

Get the environment of the atomic proposition.

**11.3.4.6** `const size_t spot::ltl::formula::hash () const` [inline, inherited]

Return a hash\_key for the formula.

**11.3.4.7** `static atomic_prop* spot::ltl::atomic_prop::instance (const std::string & name, environment & env)` [static]

Build an atomic proposition with name *name* in environment *env*.

**11.3.4.8** `static unsigned spot::ltl::atomic_prop::instance_count ()` [static]

Number of instantiated atomic propositions. For debugging.

**11.3.4.9** `const std::string& spot::ltl::atomic_prop::name () const`

Get the name of the atomic proposition.

**11.3.4.10** `formula* spot::ltl::formula::ref ()` [inherited]

clone this node

This increments the reference counter of this node (if one is used). You should almost never use this method directly as it doesn't touch the children. If you want to clone a whole formula, use `spot::ltl::clone()` instead.

**11.3.4.11** `void spot::ltl::ref_formula::ref ()` [protected, virtual, inherited]

increment reference counter if any

Reimplemented from `spot::ltl::formula`.

**11.3.4.12** `unsigned spot::ltl::ref_formula::ref_count ()` [protected, inherited]

Number of references to this formula.

**11.3.4.13** `void spot::ltl::formula::set_key_ ()` [protected, inherited]

Compute key\_ from dump\_.

Should be called once in each object, after dump\_ has been set.

**11.3.4.14** `static void spot::ltl::formula::unref (formula *f)` [static, inherited]

release this node

This decrements the reference counter of this node (if one is used) and can free the object. You should almost never use this method directly as it doesn't touch the children. If you want to release a whole formula, use `spot::ltl::destroy()` instead.

**11.3.4.15** `bool spot::ltl::ref_formula::unref ()` [protected, virtual, inherited]

decrement reference counter if any, return true when the instance must be deleted (usually when the counter hits 0).

Reimplemented from `spot::ltl::formula`.

### 11.3.5 Member Data Documentation

#### 11.3.5.1 std::string spot::ltl::formula::dump\_ [protected, inherited]

The canonic representation of the formula.

#### 11.3.5.2 environment\* spot::ltl::atomic\_prop::env\_ [private]

#### 11.3.5.3 size\_t spot::ltl::formula::hash\_key\_ [protected, inherited]

The hash key of this formula.

Initialized by [set\\_key\\_\(\)](#).

#### 11.3.5.4 map spot::ltl::atomic\_prop::instances [static, protected]

#### 11.3.5.5 std::string spot::ltl::atomic\_prop::name\_ [private]

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

- [ltlast/atomic\\_prop.hh](#)

## 11.4 spot::barand< gen > Class Template Reference

Compute pseudo-random integer value between 0 and  $n$  included, following a binomial distribution for probability  $p$ .

```
#include <misc/random.hh>
```

### Public Member Functions

- [barand](#) (int n, double p)
- int [rand](#) () const

### Protected Attributes

- const int [n\\_](#)
- const double [m\\_](#)
- const double [s\\_](#)

### 11.4.1 Detailed Description

```
template<double(*)() gen> class spot::barand< gen >
```

Compute pseudo-random integer value between 0 and  $n$  included, following a binomial distribution for probability  $p$ .

*gen* must be a random function computing a pseudo-random double value following a standard normal distribution. Use [nrnd\(\)](#) or [bmrnd\(\)](#).

Usually approximating a binomial distribution using a normal distribution and is accurate only if  $n \cdot p$  and  $n \cdot (1 - p)$  are greater than 5.

## 11.4.2 Constructor & Destructor Documentation

11.4.2.1 `template<double(*)() gen> spot::barand< gen >::barand (int n, double p)` [inline]

## 11.4.3 Member Function Documentation

11.4.3.1 `template<double(*)() gen> int spot::barand< gen >::rand () const` [inline]

## 11.4.4 Member Data Documentation

11.4.4.1 `template<double(*)() gen> const double spot::barand< gen >::m_` [protected]

11.4.4.2 `template<double(*)() gen> const int spot::barand< gen >::n_` [protected]

11.4.4.3 `template<double(*)() gen> const double spot::barand< gen >::s_` [protected]

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

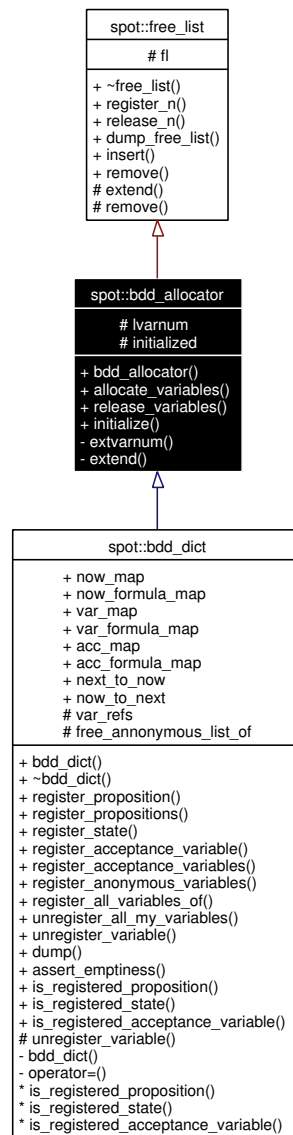
- [misc/random.hh](#)

## 11.5 spot::bdd\_allocator Class Reference

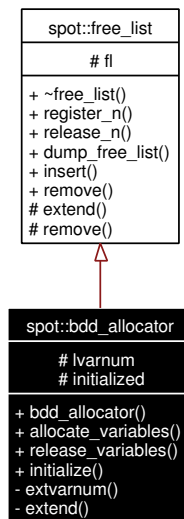
Manage ranges of variables.

```
#include <misc/bddalloc.hh>
```

Inheritance diagram for `spot::bdd_allocator`:



Collaboration diagram for `spot::bdd_allocator`:



### Public Member Functions

- [bdd\\_allocator](#) ()  
*Default constructor.*
- [allocate\\_variables](#) (int n)  
*Allocate n BDD variables.*
- [release\\_variables](#) (int base, int n)  
*Release n BDD variables starting at base.*

### Static Public Member Functions

- static void [initialize](#) ()  
*Initialize the BDD library.*

### Protected Attributes

- [lvarnum](#)  
*number of variables in use in this allocator.*

### Static Protected Attributes

- static bool [initialized](#)  
*Whether the BDD library has been initialized.*



### Private Types

- typedef std::pair< int, int > [pos\\_lenght\\_pair](#)  
*Such pairs describe second free integer starting at first.*
- typedef std::list< [pos\\_lenght\\_pair](#) > [free\\_list\\_type](#)

### Private Member Functions

- void [extvarnum](#) (int more)  
*Require more variables.*
- virtual int [extend](#) (int n)
- int [register\\_n](#) (int n)  
*Find n consecutive integers.*
- void [release\\_n](#) (int base, int n)  
*Release n consecutive integers starting at base.*
- std::ostream & [dump\\_free\\_list](#) (std::ostream &os) const  
*Dump the list to os for debugging.*
- void [insert](#) (int base, int n)  
*Extend the list by inserting a new pos-lenght pair.*
- void [remove](#) (int base, int n=0)  
*Remove n consecutive entries from the list, starting at base.*
- void [remove](#) (free\_list\_type::iterator i, int base, int n)  
*Remove n consecutive entries from the list, starting at base.*

### Private Attributes

- [free\\_list\\_type](#) fl  
*Tracks unused BDD variables.*

#### 11.5.1 Detailed Description

Manage ranges of variables.

#### 11.5.2 Member Typedef Documentation

**11.5.2.1** typedef     std::list<[pos\\_lenght\\_pair](#)>     [spot::free\\_list::free\\_list\\_type](#) [protected, inherited]

**11.5.2.2** `typedef std::pair<int, int> spot::free_list::pos_lenght_pair` [protected, inherited]

Such pairs describe second free integer starting at first.

### 11.5.3 Constructor & Destructor Documentation

**11.5.3.1** `spot::bdd_allocator::bdd_allocator ()`

Default constructor.

### 11.5.4 Member Function Documentation

**11.5.4.1** `int spot::bdd_allocator::allocate_variables (int n)`

Allocate *n* BDD variables.

**11.5.4.2** `std::ostream& spot::free_list::dump_free_list (std::ostream & os) const` [inherited]

Dump the list to *os* for debugging.

**11.5.4.3** `virtual int spot::bdd_allocator::extend (int n)` [private, virtual]

Allocate *n* integer.

This function is called by `register_n()` when the free list is empty or if *n* consecutive integers could not be found. It should allocate more integers, possibly changing the list, and return the first integer on a range of *n* consecutive integer requested by the user.

Implements `spot::free_list`.

**11.5.4.4** `void spot::bdd_allocator::extvarnum (int more)` [private]

Require more variables.

**11.5.4.5** `static void spot::bdd_allocator::initialize ()` [static]

Initialize the BDD library.

**11.5.4.6** `void spot::free_list::insert (int base, int n)` [inherited]

Extend the list by inserting a new pos-lenght pair.

**11.5.4.7** `int spot::free_list::register_n (int n)` [inherited]

Find *n* consecutive integers.

Browse the list of free integers until *n* consecutive integers are found. Extend the list (using `extend()`) otherwise.

#### Returns:

the first integer of the range

**11.5.4.8 void spot::free\_list::release\_n (int *base*, int *n*)** [inherited]

Release *n* consecutive integers starting at *base*.

**11.5.4.9 void spot::bdd\_allocator::release\_variables (int *base*, int *n*)**

Release *n* BDD variables starting at *base*.

**11.5.4.10 void spot::free\_list::remove (free\_list\_type::iterator *i*, int *base*, int *n*)** [protected, inherited]

Remove *n* consecutive entries from the list, starting at *base*.

**11.5.4.11 void spot::free\_list::remove (int *base*, int *n* = 0)** [inherited]

Remove *n* consecutive entries from the list, starting at *base*.

**11.5.5 Member Data Documentation****11.5.5.1 free\_list\_type spot::free\_list::fl** [protected, inherited]

Tracks unused BDD variables.

**11.5.5.2 bool spot::bdd\_allocator::initialized** [static, protected]

Whether the BDD library has been initialized.

**11.5.5.3 int spot::bdd\_allocator::lvarnum** [protected]

number of variables in use in this allocator.

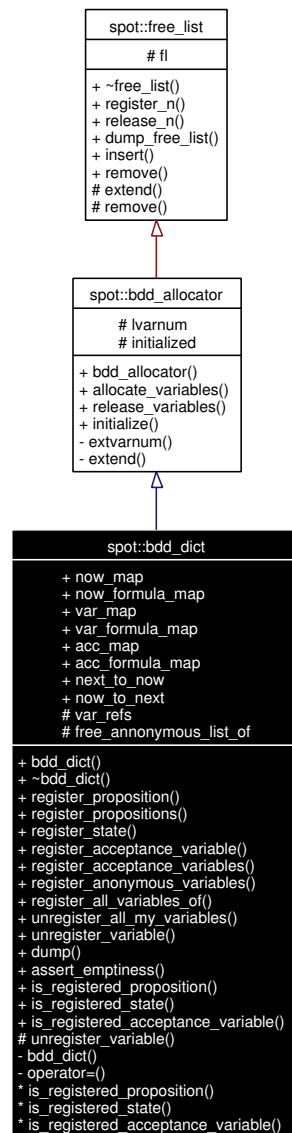
The documentation for this class was generated from the following file:

- [misc/bddalloc.hh](#)

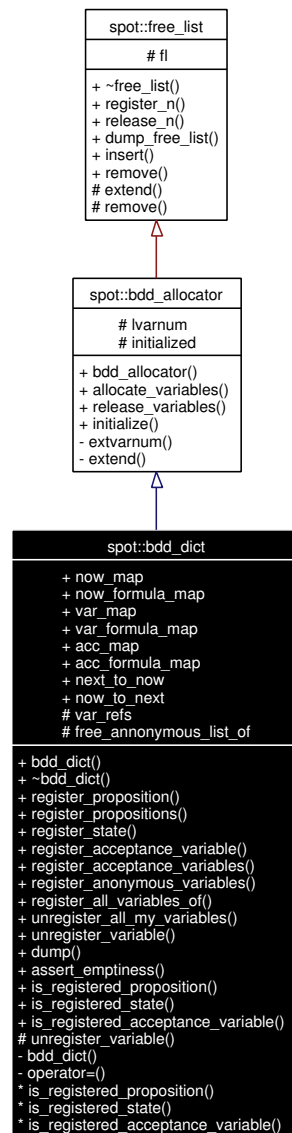
**11.6 spot::bdd\_dict Class Reference**

```
#include <tgba/bdddict.hh>
```

Inheritance diagram for spot::bdd\_dict:



Collaboration diagram for spot::bdd\_dict:



## Public Types

- typedef std::map< const [ltl::formula](#) \*, int > [fv\\_map](#)  
*Formula-to-BDD-variable maps.*
- typedef std::map< int, const [ltl::formula](#) \* > [vf\\_map](#)  
*BDD-variable-to-formula maps.*

## Public Member Functions

- [bdd\\_dict](#) ()
- [~bdd\\_dict](#) ()
- [register\\_proposition](#) (const [ltl::formula](#) \*f, const void \*for\_me)

*Register an atomic proposition.*

- void [register\\_propositions](#) (bdd f, const void \*for\_me)  
*Register BDD variables as atomic propositions.*
- int [register\\_state](#) (const [ltl::formula](#) \*f, const void \*for\_me)  
*Register a couple of Now/Next variables.*
- int [register\\_acceptance\\_variable](#) (const [ltl::formula](#) \*f, const void \*for\_me)  
*Register an atomic proposition.*
- void [register\\_acceptance\\_variables](#) (bdd f, const void \*for\_me)  
*Register BDD variables as acceptance variables.*
- int [register\\_anonymous\\_variables](#) (int n, const void \*for\_me)  
*Register anonymous BDD variables.*
- void [register\\_all\\_variables\\_of](#) (const void \*from\_other, const void \*for\_me)  
*Duplicate the variable usage of another object.*
- void [unregister\\_all\\_my\\_variables](#) (const void \*me)  
*Release all variables used by an object.*
- void [unregister\\_variable](#) (int var, const void \*me)  
*Release a variable used by me.*
- std::ostream & [dump](#) (std::ostream &os) const  
*Dump all variables for debugging.*
- void [assert\\_emptiness](#) () const  
*Make sure the dictionary is empty.*
- int [allocate\\_variables](#) (int n)  
*Allocate n BDD variables.*
- void [release\\_variables](#) (int base, int n)  
*Release n BDD variables starting at base.*
- bool [is\\_registered\\_proposition](#) (const [ltl::formula](#) \*f, const void \*by\_me)
- bool [is\\_registered\\_state](#) (const [ltl::formula](#) \*f, const void \*by\_me)
- bool [is\\_registered\\_acceptance\\_variable](#) (const [ltl::formula](#) \*f, const void \*by\_me)

### Static Public Member Functions

- static void [initialize](#) ()  
*Initialize the BDD library.*

### Public Attributes

- [fv\\_map now\\_map](#)  
*Maps formulae to "Now" BDD variables.*
- [vf\\_map now\\_formula\\_map](#)  
*Maps "Now" BDD variables to formulae.*
- [fv\\_map var\\_map](#)  
*Maps atomic propositions to BDD variables.*
- [vf\\_map var\\_formula\\_map](#)  
*Maps BDD variables to atomic propositions.*
- [fv\\_map acc\\_map](#)  
*Maps acceptance conditions to BDD variables.*
- [vf\\_map acc\\_formula\\_map](#)  
*Maps BDD variables to acceptance conditions.*
- `bddPair * next\_to\_now`  
*Map Next variables to Now variables.*
- `bddPair * now\_to\_next`  
*Map Now variables to Next variables.*

### Protected Types

- `typedef std::set< const void * > ref\_set`  
*BDD-variable reference counts.*
- `typedef std::map< int, ref\_set > vr\_map`
- `typedef std::map< const void *, annon\_free\_list > free\_anonymous\_list\_of\_type`  
*List of unused anonymous variable number for each automaton.*

### Protected Member Functions

- `void unregister\_variable (vr_map::iterator &cur, const void *me)`

### Protected Attributes

- `vr\_map var\_refs`
- `free\_anonymous\_list\_of\_type free\_anonymous\_list\_of`
- `int lvarnum`  
*number of variables in use in this allocator.*

### Static Protected Attributes

- static bool [initialized](#)

*Whether the BDD library has been initialized.*

### Private Member Functions

- [bdd\\_dict](#) (const [bdd\\_dict](#) &other)
- [bdd\\_dict](#) & [operator=](#) (const [bdd\\_dict](#) &other)

### Classes

- class [annon\\_free\\_list](#)

#### 11.6.1 Detailed Description

Map BDD variables to formulae.

#### 11.6.2 Member Typedef Documentation

**11.6.2.1** `typedef std::map<const void*, annon\_free\_list> spot::bdd\_dict::free\_anonymous\_list\_of\_type [protected]`

List of unused anonymous variable number for each automaton.

**11.6.2.2** `typedef std::map<const ltl::formula\*, int> spot::bdd\_dict::fv\_map`

Formula-to-BDD-variable maps.

**11.6.2.3** `typedef std::set<const void*> spot::bdd\_dict::ref\_set [protected]`

BDD-variable reference counts.

**11.6.2.4** `typedef std::map<int, const ltl::formula\*> spot::bdd\_dict::vf\_map`

BDD-variable-to-formula maps.

**11.6.2.5** `typedef std::map<int, ref\_set> spot::bdd\_dict::vr\_map [protected]`

#### 11.6.3 Constructor & Destructor Documentation

**11.6.3.1** `spot::bdd\_dict::bdd\_dict ()`

**11.6.3.2** `spot::bdd\_dict::~~bdd\_dict ()`

**11.6.3.3** `spot::bdd\_dict::bdd\_dict (const bdd\_dict &other) [private]`



### 11.6.4 Member Function Documentation

#### 11.6.4.1 int spot::bdd\_allocator::allocate\_variables (int *n*) [inherited]

Allocate *n* BDD variables.

#### 11.6.4.2 void spot::bdd\_dict::assert\_emptiness () const

Make sure the dictionary is empty.

This will print diagnostics and abort if the dictionary is not empty. Use for debugging.

#### 11.6.4.3 std::ostream& spot::bdd\_dict::dump (std::ostream & *os*) const

Dump all variables for debugging.

#### Parameters:

*os* The output stream.

#### 11.6.4.4 static void spot::bdd\_allocator::initialize () [static, inherited]

Initialize the BDD library.

#### 11.6.4.5 bool spot::bdd\_dict::is\_registered\_acceptance\_variable (const ltl::formula \* *f*, const void \* *by\_me*)

#### 11.6.4.6 bool spot::bdd\_dict::is\_registered\_proposition (const ltl::formula \* *f*, const void \* *by\_me*)

Check whether formula *f* has already been registered by *by\_me*.

#### 11.6.4.7 bool spot::bdd\_dict::is\_registered\_state (const ltl::formula \* *f*, const void \* *by\_me*)

#### 11.6.4.8 bdd\_dict& spot::bdd\_dict::operator= (const bdd\_dict & *other*) [private]

#### 11.6.4.9 int spot::bdd\_dict::register\_acceptance\_variable (const ltl::formula \* *f*, const void \* *for\_me*)

Register an atomic proposition.

Return (and maybe allocate) a BDD variable designating an acceptance set associated to formula *f*. The *for\_me* argument should point to the object using this BDD variable, this is used for reference counting. It is perfectly safe to call this function several time with the same arguments.

#### Returns:

The variable number. Use bdd\_ithvar() or bdd\_nithvar() to convert this to a BDD.

**11.6.4.10 void spot::bdd\_dict::register\_acceptance\_variables (bdd *f*, const void \**for\_me*)**

Register BDD variables as acceptance variables.

Register all variables occurring in *f* as acceptance variables used by *for\_me*. This assumes that these acceptance variables are already known from the dictionary (i.e., they have already been registered by [register\\_acceptance\\_variable\(\)](#) for another automaton).

**11.6.4.11 void spot::bdd\_dict::register\_all\_variables\_of (const void \**from\_other*, const void \**for\_me*)**

Duplicate the variable usage of another object.

This tells this dictionary that the *for\_me* object will be using the same BDD variables as the *from\_other* objects. This ensure that the variables won't be freed when *from\_other* is deleted if *from\_other* is still alive.

**11.6.4.12 int spot::bdd\_dict::register\_anonymous\_variables (int *n*, const void \**for\_me*)**

Register anonymous BDD variables.

Return (and maybe allocate) *n* consecutive BDD variables which will be used only by *for\_me*.

**Returns:**

The variable number. Use [bdd\\_ithvar\(\)](#) or [bdd\\_nithvar\(\)](#) to convert this to a BDD.

**11.6.4.13 int spot::bdd\_dict::register\_proposition (const [ltl::formula](#) \**f*, const void \**for\_me*)**

Register an atomic proposition.

Return (and maybe allocate) a BDD variable designating formula *f*. The *for\_me* argument should point to the object using this BDD variable, this is used for reference counting. It is perfectly safe to call this function several time with the same arguments.

**Returns:**

The variable number. Use [bdd\\_ithvar\(\)](#) or [bdd\\_nithvar\(\)](#) to convert this to a BDD.

**11.6.4.14 void spot::bdd\_dict::register\_propositions (bdd *f*, const void \**for\_me*)**

Register BDD variables as atomic propositions.

Register all variables occurring in *f* as atomic propositions used by *for\_me*. This assumes that these atomic propositions are already known from the dictionary (i.e., they have already been registered by [register\\_proposition\(\)](#) for another automaton).

**11.6.4.15 int spot::bdd\_dict::register\_state (const [ltl::formula](#) \**f*, const void \**for\_me*)**

Register a couple of Now/Next variables.

Return (and maybe allocate) two BDD variables for a state associated to formula *f*. The *for\_me* argument should point to the object using this BDD variable, this is used for reference counting. It is perfectly safe to call this function several time with the same arguments.

**Returns:**

The first variable number. Add one to get the second variable. Use [bdd\\_ithvar\(\)](#) or [bdd\\_nithvar\(\)](#) to convert this to a BDD.

**11.6.4.16** `void spot::bdd_allocator::release_variables (int base, int n)` [inherited]

Release *n* BDD variables starting at *base*.

**11.6.4.17** `void spot::bdd_dict::unregister_all_my_variables (const void * me)`

Release all variables used by an object.

Usually called in the destructor if *me*.

**11.6.4.18** `void spot::bdd_dict::unregister_variable (vr_map::iterator & cur, const void * me)`  
[protected]**11.6.4.19** `void spot::bdd_dict::unregister_variable (int var, const void * me)`

Release a variable used by *me*.

**11.6.5 Member Data Documentation****11.6.5.1** `vf_map spot::bdd_dict::acc_formula_map`

Maps BDD variables to acceptance conditions.

**11.6.5.2** `fv_map spot::bdd_dict::acc_map`

Maps acceptance conditions to BDD variables.

**11.6.5.3** `free_anonymous_list_of_type` `spot::bdd_dict::free_anonymous_list_of`  
[protected]**11.6.5.4** `bool spot::bdd_allocator::initialized` [static, protected, inherited]

Whether the BDD library has been initialized.

**11.6.5.5** `int spot::bdd_allocator::lvarnum` [protected, inherited]

number of variables in use in this allocator.

**11.6.5.6** `bddPair* spot::bdd_dict::next_to_now`

Map Next variables to Now variables.

Use with BuDDy's `bdd_replace()` function.

**11.6.5.7** `vf_map spot::bdd_dict::now_formula_map`

Maps "Now" BDD variables to formulae.

**11.6.5.8** `fv_map spot::bdd_dict::now_map`

Maps formulae to "Now" BDD variables.

**11.6.5.9 bddPair\* spot::bdd\_dict::now\_to\_next**

Map Now variables to Next variables.

Use with BuDDy's bdd\_replace() function.

**11.6.5.10 vf\_map spot::bdd\_dict::var\_formula\_map**

Maps BDD variables to atomic propositions.

**11.6.5.11 fv\_map spot::bdd\_dict::var\_map**

Maps atomic propositions to BDD variables.

**11.6.5.12 vr\_map spot::bdd\_dict::var\_refs** [protected]

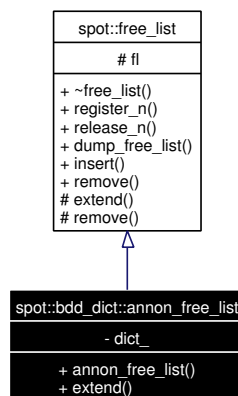
The documentation for this class was generated from the following file:

- [tgba/bdddict.hh](#)

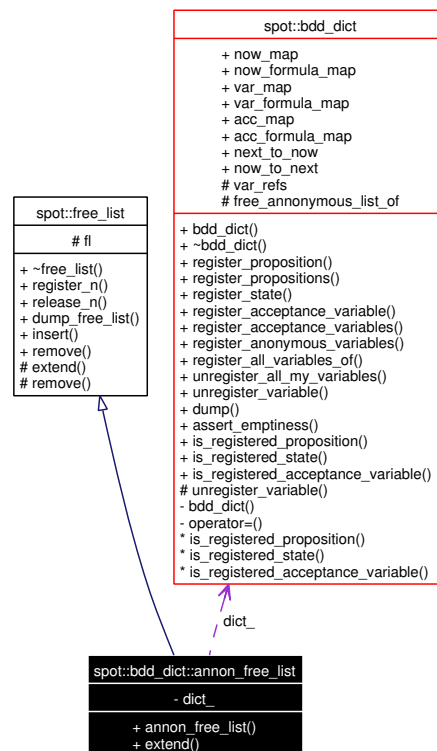
**11.7 spot::bdd\_dict::annon\_free\_list Class Reference**

```
#include <tgba/bdddict.hh>
```

Inheritance diagram for spot::bdd\_dict::annon\_free\_list:



Collaboration diagram for spot::bdd\_dict::annon\_free\_list:



## Public Member Functions

- `anon_free_list (bdd_dict *d=0)`
- `virtual int extend (int n)`
- `int register_n (int n)`  
Find *n* consecutive integers.
- `void release_n (int base, int n)`  
Release *n* consecutive integers starting at *base*.
- `std::ostream & dump_free_list (std::ostream &os) const`  
Dump the list to *os* for debugging.
- `void insert (int base, int n)`  
Extend the list by inserting a new pos-length pair.
- `void remove (int base, int n=0)`  
Remove *n* consecutive entries from the list, starting at *base*.

## Protected Types

- `typedef std::pair< int, int > pos_lenght_pair`  
Such pairs describe second free integer starting at first.
- `typedef std::list< pos_lenght_pair > free_list_type`

### Protected Member Functions

- void [remove](#) (free\_list\_type::iterator i, int base, int n)  
*Remove n consecutive entries from the list, starting at base.*

### Protected Attributes

- [free\\_list\\_type](#) fl  
*Tracks unused BDD variables.*

### Private Attributes

- [bdd\\_dict](#) \* dict\_

#### 11.7.1 Member Typedef Documentation

**11.7.1.1** typedef `std::list<pos\_lenght\_pair>` [spot::free\\_list::free\\_list\\_type](#) [protected, inherited]

**11.7.1.2** typedef `std::pair<int, int>` [spot::free\\_list::pos\\_lenght\\_pair](#) [protected, inherited]

Such pairs describe second free integer starting at first.

#### 11.7.2 Constructor & Destructor Documentation

**11.7.2.1** `spot::bdd_dict::annon_free_list::annon_free_list` ([bdd\\_dict](#) \* d = 0)

#### 11.7.3 Member Function Documentation

**11.7.3.1** `std::ostream& spot::free_list::dump_free_list` (std::ostream & os) const [inherited]

Dump the list to os for debugging.

**11.7.3.2** `virtual int spot::bdd_dict::annon_free_list::extend` (int n) [virtual]

Allocate n integer.

This function is called by [register\\_n\(\)](#) when the free list is empty or if n consecutive integers could not be found. It should allocate more integers, possibly changing the list, and return the first integer on a range of n consecutive integer requested by the user.

Implements [spot::free\\_list](#).

**11.7.3.3** `void spot::free_list::insert` (int base, int n) [inherited]

Extend the list by inserting a new pos-lenght pair.

**11.7.3.4 int spot::free\_list::register\_n (int *n*)** [inherited]

Find *n* consecutive integers.

Browse the list of free integers until *n* consecutive integers are found. Extend the list (using [extend\(\)](#)) otherwise.

**Returns:**

the first integer of the range

**11.7.3.5 void spot::free\_list::release\_n (int *base*, int *n*)** [inherited]

Release *n* consecutive integers starting at *base*.

**11.7.3.6 void spot::free\_list::remove (free\_list\_type::iterator *i*, int *base*, int *n*)** [protected, inherited]

Remove *n* consecutive entries from the list, starting at *base*.

**11.7.3.7 void spot::free\_list::remove (int *base*, int *n* = 0)** [inherited]

Remove *n* consecutive entries from the list, starting at *base*.

**11.7.4 Member Data Documentation****11.7.4.1 bdd\_dict\* spot::bdd\_dict::annon\_free\_list::dict\_** [private]**11.7.4.2 free\_list\_type spot::free\_list::fl** [protected, inherited]

Tracks unused BDD variables.

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

- [tgb/bdddict.hh](#)

**11.8 spot::bdd\_less\_than Struct Reference**

Comparison functor for BDDs.

```
#include <misc/bddlt.hh>
```

**Public Member Functions**

- bool [operator\(\)](#) (const bdd &left, const bdd &right) const

**11.8.1 Detailed Description**

Comparison functor for BDDs.

## 11.8.2 Member Function Documentation

**11.8.2.1** `bool spot::bdd_less_than::operator() (const bdd & left, const bdd & right) const` `[inline]`

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

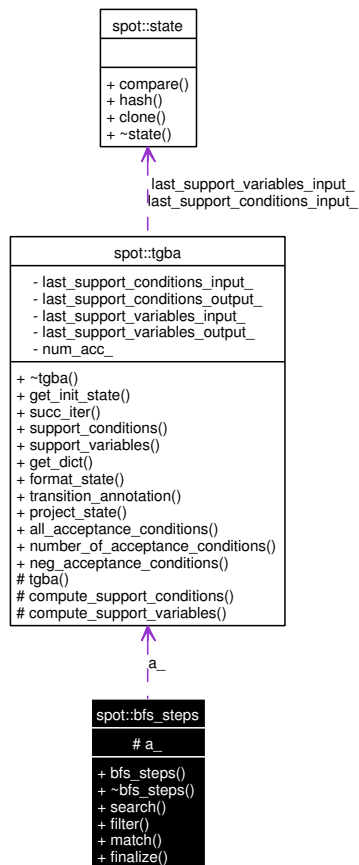
- [misc/bddlt.hh](#)

## 11.9 spot::bfs\_steps Class Reference

Make a BFS in a `spot::tgba` to compute a `tgba_run::steps`.

```
#include <tgbaalgos/bfssteps.hh>
```

Collaboration diagram for `spot::bfs_steps`:



### Public Member Functions

- `bfs_steps` (const `tgba` \*a)
- virtual `~bfs_steps` ()
- const `state` \* `search` (const `state` \*start, `tgba_run::steps` &l)

*Start the search from start, and append the resulting path (if any) to l.*



- virtual const `state*` `filter` (const `state*` `s`)=0  
*Return a state\* that is unique for a.*
- virtual bool `match` (`tgba_run::step` &`step`, const `state*` `dest`)=0  
*Whether a new transition completes a path.*
- virtual void `finalize` (const std::map< const `state*`, `tgba_run::step`, `state_ptr_less_than` > &`father`, const `tgba_run::step` &`s`, const `state*` `start`, `tgba_run::steps` &`l`)  
*Append the resulting path to the resulting run.*

### Protected Attributes

- const `tgba*` `a_`  
*The `spot::tgba` we are searching into.*

### 11.9.1 Detailed Description

Make a BFS in a `spot::tgba` to compute a `tgba_run::steps`.

This class should be used to compute the shortest path between a state of a `spot::tgba` and the first transition or state that matches some conditions.

These conditions should be specified by defining `bfs_steps::match()` in a subclass. Also the search can be restricted to some set of states with a proper definition of `bfs_steps::filter()`.

### 11.9.2 Constructor & Destructor Documentation

#### 11.9.2.1 spot::bfs\_steps::bfs\_steps (const `tgba*` `a`)

#### 11.9.2.2 virtual spot::bfs\_steps::~bfs\_steps () [virtual]

### 11.9.3 Member Function Documentation

#### 11.9.3.1 virtual const `state*` spot::bfs\_steps::filter (const `state*` `s`) [pure virtual]

Return a state\* that is unique for `a`.

`bfs_steps` does not do handle the memory for the states it generates, this is the job of `filter()`. Here `s` is a new state\* that `search()` has just allocated (using `tgba_succ_iterator::current_state()`), and the return of this function should be a state\* that does not need to be freed by `search()`.

If you already have a map or a set which uses states as keys, you should probably arrange for `filter()` to return these keys, and delete `s`. Otherwise you will have to define such a set, just to be able to delete all the state\* in a subclass.

This function can return 0 if the given state should not be explored.

**11.9.3.2** `virtual void spot::bfs_steps::finalize (const std::map< const state *, tgba\_run::step, state\_ptr\_less\_than > & father, const tgba\_run::step & s, const state * start, tgba\_run::steps & l)` `[virtual]`

Append the resulting path to the resulting run.

This is called after [match\(\)](#) has returned true, to append the resulting path to *l*. This seldom needs to be overridden, unless you do not want *l* to be updated (in which case an empty [finalize\(\)](#) will do).

**11.9.3.3** `virtual bool spot::bfs_steps::match (tgba\_run::step & step, const state * dest)` `[pure virtual]`

Whether a new transition completes a path.

This function is called immediately after each call to [filter\(\)](#) that does not return 0.

**Parameters:**

*step* the source state (as returned by [filter\(\)](#)), and the labels of the outgoing transition

*dest* the destination state (as returned by [filter\(\)](#))

**Returns:**

true iff a path that included this step should be accepted.

The [search\(\)](#) algorithms stops as soon as [match\(\)](#) returns true, and when this happens the list argument of [search\(\)](#) is be augmented with the shortest past that ends with this transition.

**11.9.3.4** `const state* spot::bfs_steps::search (const state * start, tgba\_run::steps & l)`

Start the search from *start*, and append the resulting path (if any) to *l*.

**Returns:**

the destination state of the last step (not included in *l*) if a matching path was found, or 0 otherwise.

## 11.9.4 Member Data Documentation

**11.9.4.1** `const tgba* spot::bfs_steps::a_` `[protected]`

The [spot::tgba](#) we are searching into.

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

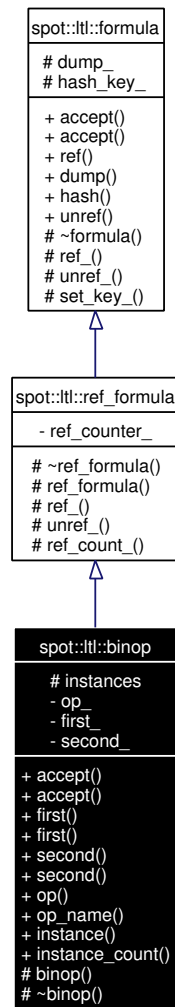
- [tgbaalgos/bfssteps.hh](#)

## 11.10 spot::ltl::binop Class Reference

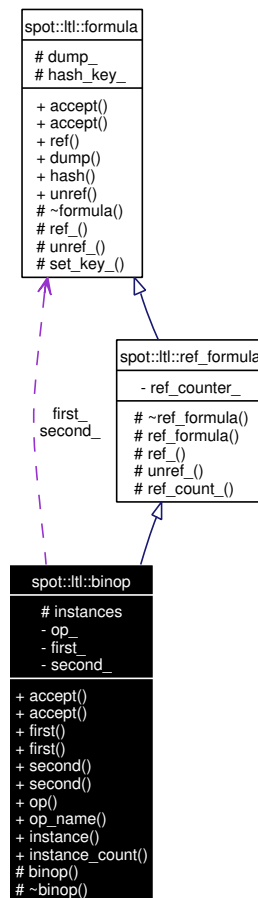
Binary operator.

```
#include <ltlast/binop.hh>
```

Inheritance diagram for `spot::ltl::binop`:



Collaboration diagram for `spot::ltl::binop`:



## Public Types

- enum `type` {  
`Xor`, `Implies`, `Equiv`, `U`,  
`R` }

## Public Member Functions

- virtual void `accept (visitor &v)`  
*Entry point for `vspot::ltl::visitor` instances.*
- virtual void `accept (const_visitor &v) const`  
*Entry point for `vspot::ltl::const_visitor` instances.*
- const `formula * first () const`  
*Get the first operand.*
- `formula * first ()`  
*Get the first operand.*

- `const formula * second () const`  
*Get the second operand.*
- `formula * second ()`  
*Get the second operand.*
- `type op () const`  
*Get the type of this operator.*
- `const char * op_name () const`  
*Get the type of this operator, as a string.*
- `formula * ref ()`  
*clone this node*
- `const std::string & dump () const`  
*Return a canonic representation of the formula.*
- `const size_t hash () const`  
*Return a hash\_key for the formula.*

### Static Public Member Functions

- `static binop * instance (type op, formula *first, formula *second)`
- `static unsigned instance_count ()`  
*Number of instantiated binary operators. For debugging.*
- `static void unref (formula *f)`  
*release this node*

### Protected Types

- `typedef std::pair< formula *, formula * > pairf`
- `typedef std::pair< type, pairf > pair`
- `typedef std::map< pair, formula * > map`

### Protected Member Functions

- `binop (type op, formula *first, formula *second)`
- `virtual ~binop ()`
- `void ref_ ()`  
*increment reference counter if any*
- `bool unref_ ()`  
*decrement reference counter if any, return true when the instance must be deleted (usually when the counter hits 0).*

- unsigned [ref\\_count\\_](#) ()  
*Number of references to this formula.*
- void [set\\_key\\_](#) ()  
*Compute key\_from dump\_.*

### Protected Attributes

- std::string [dump\\_](#)  
*The canonic representation of the formula.*
- size\_t [hash\\_key\\_](#)  
*The hash key of this formula.*

### Static Protected Attributes

- static [map instances](#)

### Private Attributes

- [type op\\_](#)
- [formula \\* first\\_](#)
- [formula \\* second\\_](#)

#### 11.10.1 Detailed Description

Binary operator.

#### 11.10.2 Member Typedef Documentation

**11.10.2.1** `typedef std::map<pair, formula\*> spot::ltl::binop::map` [protected]

**11.10.2.2** `typedef std::pair<type, pairf> spot::ltl::binop::pair` [protected]

**11.10.2.3** `typedef std::pair<formula\*, formula\*> spot::ltl::binop::pairf` [protected]

#### 11.10.3 Member Enumeration Documentation

**11.10.3.1** `enum spot::ltl::binop::type`

Different kinds of binary opertaors

And and Or are not here. Because they are often nested we represent them as multops.

#### Enumeration values:

*Xor*

*Implies**Equiv**U**R*

### 11.10.4 Constructor & Destructor Documentation

**11.10.4.1** `spot::ltl::binop::binop (type op, formula * first, formula * second)` [protected]

**11.10.4.2** `virtual spot::ltl::binop::~~binop ()` [protected, virtual]

### 11.10.5 Member Function Documentation

**11.10.5.1** `virtual void spot::ltl::binop::accept (const_visitor & v) const` [virtual]

Entry point for vspot::ltl::const\_visitor instances.

Implements `spot::ltl::formula`.

**11.10.5.2** `virtual void spot::ltl::binop::accept (visitor & v)` [virtual]

Entry point for vspot::ltl::visitor instances.

Implements `spot::ltl::formula`.

**11.10.5.3** `const std::string& spot::ltl::formula::dump () const` [inherited]

Return a canonic representation of the formula.

**11.10.5.4** `formula* spot::ltl::binop::first ()`

Get the first operand.

**11.10.5.5** `const formula* spot::ltl::binop::first () const`

Get the first operand.

**11.10.5.6** `const size_t spot::ltl::formula::hash () const` [inline, inherited]

Return a hash\_key for the formula.

**11.10.5.7** `static binop* spot::ltl::binop::instance (type op, formula * first, formula * second)`  
[static]

Build an unary operator with operation *op* and children *first* and *second*.

**11.10.5.8** `static unsigned spot::ltl::binop::instance_count ()` [static]

Number of instantiated binary operators. For debugging.

**11.10.5.9** `type` `spot::ltl::binop::op () const`

Get the type of this operator.

**11.10.5.10** `const char*` `spot::ltl::binop::op_name () const`

Get the type of this operator, as a string.

**11.10.5.11** `formula*` `spot::ltl::formula::ref ()` [inherited]

clone this node

This increments the reference counter of this node (if one is used). You should almost never use this method directly as it doesn't touch the children. If you want to clone a whole formula, use `spot::ltl::clone()` instead.

**11.10.5.12** `void` `spot::ltl::ref_formula::ref_ ()` [protected, virtual, inherited]

increment reference counter if any

Reimplemented from `spot::ltl::formula`.

**11.10.5.13** `unsigned` `spot::ltl::ref_formula::ref_count_ ()` [protected, inherited]

Number of references to this formula.

**11.10.5.14** `formula*` `spot::ltl::binop::second ()`

Get the second operand.

**11.10.5.15** `const formula*` `spot::ltl::binop::second () const`

Get the second operand.

**11.10.5.16** `void` `spot::ltl::formula::set_key_ ()` [protected, inherited]

Compute key\_ from dump\_.

Should be called once in each object, after dump\_ has been set.

**11.10.5.17** `static void` `spot::ltl::formula::unref (formula *f)` [static, inherited]

release this node

This decrements the reference counter of this node (if one is used) and can free the object. You should almost never use this method directly as it doesn't touch the children. If you want to release a whole formula, use `spot::ltl::destroy()` instead.

**11.10.5.18** `bool` `spot::ltl::ref_formula::unref_ ()` [protected, virtual, inherited]

decrement reference counter if any, return true when the instance must be deleted (usually when the counter hits 0).

Reimplemented from `spot::ltl::formula`.



### 11.10.6 Member Data Documentation

#### 11.10.6.1 std::string spot::ltl::formula::dump\_ [protected, inherited]

The canonic representation of the formula.

#### 11.10.6.2 formula\* spot::ltl::binop::first\_ [private]

#### 11.10.6.3 size\_t spot::ltl::formula::hash\_key\_ [protected, inherited]

The hash key of this formula.

Initialized by [set\\_key\\_\(\)](#).

#### 11.10.6.4 map spot::ltl::binop::instances [static, protected]

#### 11.10.6.5 type spot::ltl::binop::op\_ [private]

#### 11.10.6.6 formula\* spot::ltl::binop::second\_ [private]

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

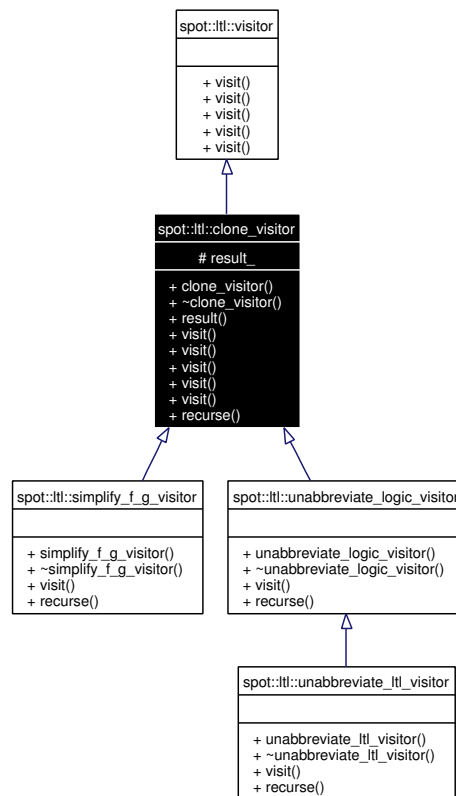
- [ltlast/binop.hh](#)

## 11.11 spot::ltl::clone\_visitor Class Reference

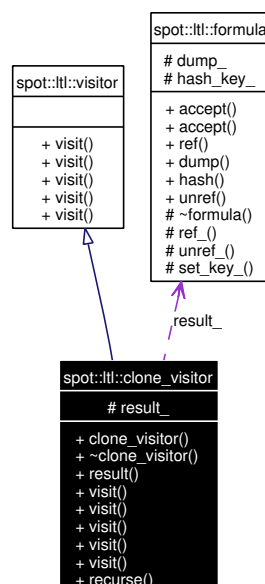
Clone a formula.

```
#include <ltlvisit/clone.hh>
```

Inheritance diagram for spot::ltl::clone\_visitor:



Collaboration diagram for `spot::ltl::clone_visitor`:



## Public Member Functions

- [clone\\_visitor](#) ()
- virtual [~clone\\_visitor](#) ()
- [formula](#) \* [result](#) () const
- void [visit](#) ([atomic\\_prop](#) \*ap)
- void [visit](#) ([unop](#) \*uo)
- void [visit](#) ([binop](#) \*bo)
- void [visit](#) ([multop](#) \*mo)
- void [visit](#) ([constant](#) \*c)
- virtual [formula](#) \* [recurse](#) ([formula](#) \*f)

## Protected Attributes

- [formula](#) \* [result\\_](#)

### 11.11.1 Detailed Description

Clone a formula.

This visitor is public, because it's convenient to derive from it and override part of its methods. But if you just want the functionality, consider using [spot::ltl::clone](#) instead.

### 11.11.2 Constructor & Destructor Documentation

#### 11.11.2.1 spot::ltl::clone\_visitor::clone\_visitor ()

#### 11.11.2.2 virtual spot::ltl::clone\_visitor::~~clone\_visitor () [virtual]

### 11.11.3 Member Function Documentation

#### 11.11.3.1 virtual [formula](#)\* spot::ltl::clone\_visitor::recurse ([formula](#) \*f) [virtual]

Reimplemented in [spot::ltl::unabbreviate\\_logic\\_visitor](#), [spot::ltl::simplify\\_f\\_g\\_visitor](#), and [spot::ltl::unabbreviate\\_ltl\\_visitor](#).

#### 11.11.3.2 [formula](#)\* spot::ltl::clone\_visitor::result () const

#### 11.11.3.3 void spot::ltl::clone\_visitor::visit ([constant](#) \*c) [virtual]

Implements [spot::ltl::visitor](#).

#### 11.11.3.4 void spot::ltl::clone\_visitor::visit ([multop](#) \*mo) [virtual]

Implements [spot::ltl::visitor](#).

#### 11.11.3.5 void spot::ltl::clone\_visitor::visit ([binop](#) \*bo) [virtual]

Implements [spot::ltl::visitor](#).

Reimplemented in [spot::ltl::unabbreviate\\_logic\\_visitor](#), and [spot::ltl::simplify\\_f\\_g\\_visitor](#).

**11.11.3.6** void `spot::ltl::clone_visitor::visit` (`unop *uo`) [virtual]

Implements `spot::ltl::visitor`.

Reimplemented in `spot::ltl::unabbreviate_ltl_visitor`.

**11.11.3.7** void `spot::ltl::clone_visitor::visit` (`atomic_prop *ap`) [virtual]

Implements `spot::ltl::visitor`.

#### 11.11.4 Member Data Documentation

**11.11.4.1** `formula*` `spot::ltl::clone_visitor::result_` [protected]

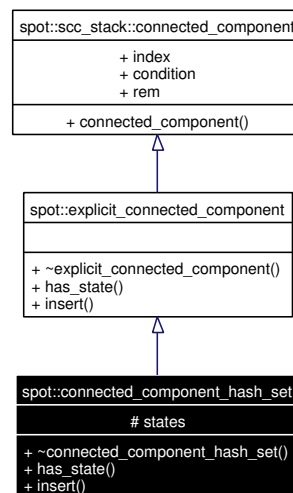
The documentation for this class was generated from the following file:

- `ltlvisit/clone.hh`

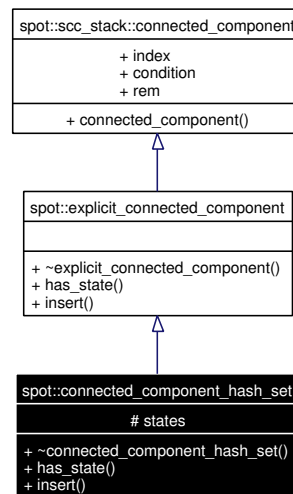
## 11.12 spot::connected\_component\_hash\_set Class Reference

```
#include <tgbaalgorithms/gtec/explsc.h>
```

Inheritance diagram for `spot::connected_component_hash_set`:



Collaboration diagram for `spot::connected_component_hash_set`:



### Public Member Functions

- virtual [~connected\\_component\\_hash\\_set](#) ()
- virtual const [state](#) \* [has\\_state](#) (const [state](#) \*s) const  
*Check if the SCC contains states s.*
- virtual void [insert](#) (const [state](#) \*s)  
*Insert a new state in the SCC.*

### Public Attributes

- int [index](#)  
*Index of the SCC.*
- bdd [condition](#)
- std::list< const [state](#) \* > [rem](#)

### Protected Types

- typedef Sgi::hash\_set< const [state](#) \*, [state\\_ptr\\_hash](#), [state\\_ptr\\_equal](#) > [set\\_type](#)

### Protected Attributes

- [set\\_type](#) [states](#)

#### 11.12.1 Detailed Description

A straightforward implementation of [explicit\\_connected\\_component](#) using a hash.

### 11.12.2 Member Typedef Documentation

**11.12.2.1** `typedef Sgi::hash_set<const state\*, state\_ptr\_hash, state\_ptr\_equal> spot::connected\_component\_hash\_set::set\_type` [protected]

### 11.12.3 Constructor & Destructor Documentation

**11.12.3.1** `virtual spot::connected\_component\_hash\_set::~~connected\_component\_hash\_set ()` [inline, virtual]

### 11.12.4 Member Function Documentation

**11.12.4.1** `virtual const state\* spot::connected\_component\_hash\_set::has\_state (const state * s) const` [virtual]

Check if the SCC contains states *s*.

Return the representative of *s* in the SCC, and delete *s* if it is different (acting like `numbered_state_heap::filter`), or 0 otherwise.

Implements [spot::explicit\\_connected\\_component](#).

**11.12.4.2** `virtual void spot::connected\_component\_hash\_set::insert (const state * s)` [virtual]

Insert a new state in the SCC.

Implements [spot::explicit\\_connected\\_component](#).

### 11.12.5 Member Data Documentation

**11.12.5.1** `bdd spot::scc\_stack::connected\_component::condition` [inherited]

The bdd condition is the union of all acceptance conditions of transitions which connect the states of the connected component.

**11.12.5.2** `int spot::scc\_stack::connected\_component::index` [inherited]

Index of the SCC.

**11.12.5.3** `std::list<const state\*> spot::scc\_stack::connected\_component::rem` [inherited]

**11.12.5.4** `set\_type spot::connected\_component\_hash\_set::states` [protected]

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

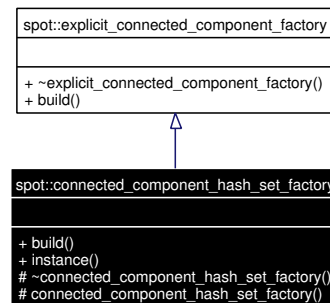
- [tgbaalgos/gtec/explscg.hh](#)

## 11.13 spot::connected\_component\_hash\_set\_factory Class Reference

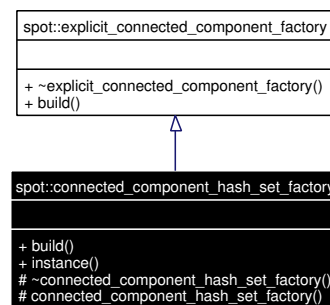
Factory for [connected\\_component\\_hash\\_set](#).

```
#include <tgbaalgos/gtec/explscg.hh>
```

Inheritance diagram for spot::connected\_component\_hash\_set\_factory:



Collaboration diagram for spot::connected\_component\_hash\_set\_factory:



### Public Member Functions

- virtual `connected_component_hash_set * build ()` const  
Create an *explicit\_connected\_component*.

### Static Public Member Functions

- static const `connected_component_hash_set_factory * instance ()`  
Get the unique instance of this class.

### Protected Member Functions

- virtual `~connected_component_hash_set_factory ()`
- `connected_component_hash_set_factory ()`  
Construction is forbidden.

#### 11.13.1 Detailed Description

Factory for `connected_component_hash_set`.

This class is a singleton. Retrieve the instance using [instance\(\)](#).

### 11.13.2 Constructor & Destructor Documentation

**11.13.2.1** virtual [spot::connected\\_component\\_hash\\_set\\_factory::~~connected\\_component\\_hash\\_set\\_factory](#) () [inline, protected, virtual]

**11.13.2.2** [spot::connected\\_component\\_hash\\_set\\_factory::connected\\_component\\_hash\\_set\\_factory](#) () [protected]

Construction is forbidden.

### 11.13.3 Member Function Documentation

**11.13.3.1** virtual [connected\\_component\\_hash\\_set\\*](#) [spot::connected\\_component\\_hash\\_set\\_factory::build](#) () const [virtual]

Create an [explicit\\_connected\\_component](#).

Implements [spot::explicit\\_connected\\_component\\_factory](#).

**11.13.3.2** static const [connected\\_component\\_hash\\_set\\_factory\\*](#) [spot::connected\\_component\\_hash\\_set\\_factory::instance](#) () [static]

Get the unique instance of this class.

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

- [tgbaalgos/gtec/explsc.hh](#)

## 11.14 spot::ltl::const\_visitor Struct Reference

Formula visitor that cannot modify the formula.

```
#include <ltlast/visitor.hh>
```

### Public Member Functions

- virtual void [visit](#) (const [atomic\\_prop](#) \*node)=0
- virtual void [visit](#) (const [constant](#) \*node)=0
- virtual void [visit](#) (const [binop](#) \*node)=0
- virtual void [visit](#) (const [unop](#) \*node)=0
- virtual void [visit](#) (const [multop](#) \*node)=0

### 11.14.1 Detailed Description

Formula visitor that cannot modify the formula.

Writing visitors is the preferred way to traverse a formula, since it doesn't involve any cast.

If you want to modify the visited formula, inherit from [spot::ltl:visitor](#) instead.



### 11.14.2 Member Function Documentation

**11.14.2.1** virtual void spot::ltl::const\_visitor::visit (const **multop** \* *node*) [pure virtual]

**11.14.2.2** virtual void spot::ltl::const\_visitor::visit (const **unop** \* *node*) [pure virtual]

**11.14.2.3** virtual void spot::ltl::const\_visitor::visit (const **binop** \* *node*) [pure virtual]

**11.14.2.4** virtual void spot::ltl::const\_visitor::visit (const **constant** \* *node*) [pure virtual]

**11.14.2.5** virtual void spot::ltl::const\_visitor::visit (const **atomic\_prop** \* *node*) [pure virtual]

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

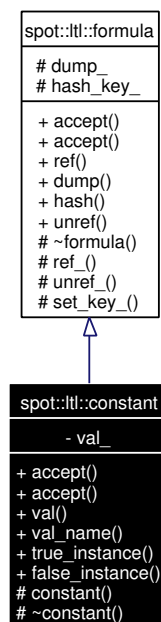
- ltlast/[visitor.hh](#)

## 11.15 spot::ltl::constant Class Reference

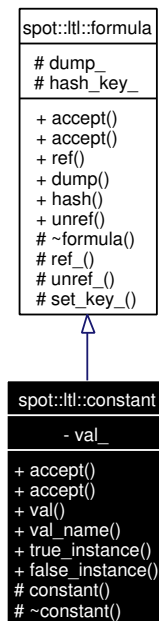
A constant (True or False).

```
#include <ltlast/constant.hh>
```

Inheritance diagram for spot::ltl::constant:



Collaboration diagram for spot::ltl::constant:



## Public Types

- enum `type` { `False`, `True` }

## Public Member Functions

- virtual void `accept` (`visitor` &`v`)  
*Entry point for `vspot::ltl::visitor` instances.*
- virtual void `accept` (`const_visitor` &`v`) const  
*Entry point for `vspot::ltl::const_visitor` instances.*
- `type val` () const  
*Return the value of the constant.*
- const char \* `val_name` () const  
*Return the value of the constant as a string.*
- `formula` \* `ref` ()  
*clone this node*
- const std::string & `dump` () const  
*Return a canonic representation of the formula.*
- const size\_t `hash` () const  
*Return a hash\_key for the formula.*

### Static Public Member Functions

- static [constant](#) \* [true\\_instance](#) ()  
*Get the sole instance of spot::lfl::constant::constant(True).*
- static [constant](#) \* [false\\_instance](#) ()  
*Get the sole instance of spot::lfl::constant::constant(False).*
- static void [unref](#) ([formula](#) \*f)  
*release this node*

### Protected Member Functions

- [constant](#) (type val)
- virtual [~constant](#) ()
- virtual void [ref\\_](#) ()  
*increment reference counter if any*
- virtual bool [unref\\_](#) ()  
*decrement reference counter if any, return true when the instance must be deleted (usually when the counter hits 0).*
- void [set\\_key\\_](#) ()  
*Compute key\_ from dump\_.*

### Protected Attributes

- std::string [dump\\_](#)  
*The canonic representation of the formula.*
- size\_t [hash\\_key\\_](#)  
*The hash key of this formula.*

### Private Attributes

- type [val\\_](#)

#### 11.15.1 Detailed Description

A constant (True or False).

#### 11.15.2 Member Enumeration Documentation

##### 11.15.2.1 enum [spot::lfl::constant::type](#)

Enumeration values:

*False*

*True*

### 11.15.3 Constructor & Destructor Documentation

**11.15.3.1** `spot::ltl::constant::constant (type val)` [protected]

**11.15.3.2** `virtual spot::ltl::constant::~~constant ()` [protected, virtual]

### 11.15.4 Member Function Documentation

**11.15.4.1** `virtual void spot::ltl::constant::accept (const_visitor & v) const` [virtual]

Entry point for `vspot::ltl::const_visitor` instances.

Implements [spot::ltl::formula](#).

**11.15.4.2** `virtual void spot::ltl::constant::accept (visitor & v)` [virtual]

Entry point for `vspot::ltl::visitor` instances.

Implements [spot::ltl::formula](#).

**11.15.4.3** `const std::string& spot::ltl::formula::dump () const` [inherited]

Return a canonic representation of the formula.

**11.15.4.4** `static constant* spot::ltl::constant::false_instance ()` [static]

Get the sole instance of `spot::ltl::constant::constant(False)`.

**11.15.4.5** `const size_t spot::ltl::formula::hash () const` [inline, inherited]

Return a `hash_key` for the formula.

**11.15.4.6** `formula* spot::ltl::formula::ref ()` [inherited]

clone this node

This increments the reference counter of this node (if one is used). You should almost never use this method directly as it doesn't touch the children. If you want to clone a whole formula, use [spot::ltl::clone\(\)](#) instead.

**11.15.4.7** `virtual void spot::ltl::formula::ref_ ()` [protected, virtual, inherited]

increment reference counter if any

Reimplemented in [spot::ltl::ref\\_formula](#).

**11.15.4.8** `void spot::ltl::formula::set_key_ ()` [protected, inherited]

Compute `key_` from `dump_`.

Should be called once in each object, after `dump_` has been set.

**11.15.4.9** `static constant* spot::ltl::constant::true_instance ()` [static]

Get the sole instance of `spot::ltl::constant::constant(True)`.

**11.15.4.10 static void spot::ltl::formula::unref (formula \*f) [static, inherited]**

release this node

This decrements the reference counter of this node (if one is used) and can free the object. You should almost never use this method directly as it doesn't touch the children. If you want to release a whole formula, use [spot::ltl::destroy\(\)](#) instead.

**11.15.4.11 virtual bool spot::ltl::formula::unref\_ () [protected, virtual, inherited]**

decrement reference counter if any, return true when the instance must be deleted (usually when the counter hits 0).

Reimplemented in [spot::ltl::ref\\_formula](#).

**11.15.4.12 type spot::ltl::constant::val () const**

Return the value of the constant.

**11.15.4.13 const char\* spot::ltl::constant::val\_name () const**

Return the value of the constant as a string.

**11.15.5 Member Data Documentation****11.15.5.1 std::string spot::ltl::formula::dump\_ [protected, inherited]**

The canonic representation of the formula.

**11.15.5.2 size\_t spot::ltl::formula::hash\_key\_ [protected, inherited]**

The hash key of this formula.

Initialized by [set\\_key\\_\(\)](#).

**11.15.5.3 type spot::ltl::constant::val\_ [private]**

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

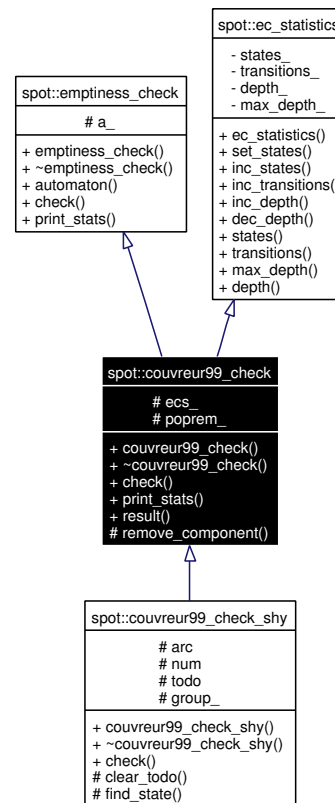
- [ltlast/constant.hh](#)

**11.16 spot::couvreur99\_check Class Reference**

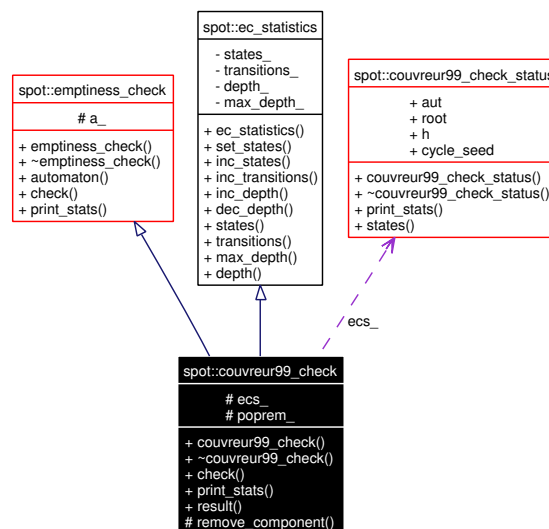
Check whether the language of an automate is empty.

```
#include <tgbalgorithms/gtec/gtec.hh>
```

Inheritance diagram for `spot::couvreur99_check`:



Collaboration diagram for spot::couvreur99\_check:



## Public Member Functions

- `couvreur99_check` (const `tgba` \*a, bool poprem=true, const `numbered_state_heap_factory` \*nshf=numbered\_state\_heap\_hash\_map\_factory::instance())

- virtual `~couvreur99_check()`
- virtual `emptiness_check_result * check()`  
*Check whether the automaton's language is empty.*
- virtual `std::ostream & print_stats(std::ostream &os) const`  
*Print statistics, if any.*
- const `couvreur99_check_status * result()` const  
*Return the status of the emptiness-check.*
- const `tgba * automaton()` const  
*The automaton that this emptiness-check inspects.*
- void `set_states(unsigned n)`
- void `inc_states()`
- void `inc_transitions()`
- void `inc_depth(unsigned n=1)`
- void `dec_depth(unsigned n=1)`
- int `states()` const
- int `transitions()` const
- int `max_depth()` const
- int `depth()` const

### Protected Member Functions

- void `remove_component(const state *start_delete)`  
*Remove a strongly component from the hash.*

### Protected Attributes

- `couvreur99_check_status * ecs_`
- bool `poprem_`  
*Whether to store the state to be removed.*
- const `tgba * a_`  
*The automaton.*

#### 11.16.1 Detailed Description

Check whether the language of an automate is empty.

This is based on the following paper.

```
@InProceedings{couvreur.99.fm,
  author    = {Jean-Michel Couvreur},
  title     = {On-the-fly Verification of Temporal Logic},
  pages     = {253--271},
  editor    = {Jeannette M. Wing and Jim Woodcock and Jim Davies},
  booktitle = {Proceedings of the World Congress on Formal Methods in
```

```

        the Development of Computing Systems (FM'99)),
    publisher = {Springer-Verlag},
    series    = {Lecture Notes in Computer Science},
    volume    = {1708},
    year      = {1999},
    address   = {Toulouse, France},
    month     = {September},
    isbn      = {3-540-66587-0}
}

```

[check\(\)](#) returns true if the automaton's language is empty. When it return false, a stack of SCC has been built is available using [result\(\)](#) (spot::counter\_example needs it).

There are two variants of this algorithm: [spot::couvreur99\\_check](#) and [spot::couvreur99\\_check\\_shy](#). They differ in their memory usage, the number for successors computed before they are used and the way the depth first search is directed.

[spot::couvreur99\\_check](#) performs a straightforward depth first search. The DFS stacks store `tgba_succ_` iterators, so that only the iterators which really are explored are computed.

[spot::couvreur99\\_check\\_shy](#) tries to explore successors which are visited states first. this helps to merge SCCs and generally helps to produce shorter counter-examples. However this algorithm cannot stores unprocessed successors as `tgba_succ_iterators`: it must compute all successors of a state at once in order to decide which to explore first, and must keep a list of all unexplored successors in its DFS stack.

The `poprem` parameter specifies how the algorithm should handle the destruction of non-accepting maximal strongly connected components. If `poprem` is true, the algorithm will keep a list of all states of a SCC that are fully processed and should be removed once the MSCC is popped. If `poprem` is false, the MSCC will be traversed again (i.e. generating the successors of the root recursively) for deletion. This is a choice between memory and speed.

## 11.16.2 Constructor & Destructor Documentation

**11.16.2.1** `spot::couvreur99_check::couvreur99_check (const tgba * a, bool poprem = true, const numbered\_state\_heap\_factory * nshf = numbered_state_heap_hash_map_factory::instance())`

**11.16.2.2** `virtual spot::couvreur99_check::~~couvreur99_check ()` [virtual]

## 11.16.3 Member Function Documentation

**11.16.3.1** `const tgba* spot::emptiness_check::automaton () const` [inline, inherited]

The automaton that this emptiness-check inspects.

**11.16.3.2** `virtual emptiness\_check\_result* spot::couvreur99_check::check ()` [virtual]

Check whether the automaton's language is empty.

Implements [spot::emptiness\\_check](#).

Reimplemented in [spot::couvreur99\\_check\\_shy](#).

**11.16.3.3** `void spot::ec_statistics::dec_depth (unsigned n = 1)` [inline, inherited]



**11.16.3.4** `int spot::ec_statistics::depth () const` [inline, inherited]

**11.16.3.5** `void spot::ec_statistics::inc_depth (unsigned n = 1)` [inline, inherited]

**11.16.3.6** `void spot::ec_statistics::inc_states ()` [inline, inherited]

**11.16.3.7** `void spot::ec_statistics::inc_transitions ()` [inline, inherited]

**11.16.3.8** `int spot::ec_statistics::max_depth () const` [inline, inherited]

**11.16.3.9** `virtual std::ostream& spot::couvreur99_check::print_stats (std::ostream & os) const`  
[virtual]

Print statistics, if any.

Reimplemented from [spot::emptiness\\_check](#).

**11.16.3.10** `void spot::couvreur99_check::remove_component (const state * start_delete)`  
[protected]

Remove a strongly component from the hash.

This function remove all accessible state from a given state. In other words, it removes the strongly connected component that contains this state.

**11.16.3.11** `const couvreur99\_check\_status* spot::couvreur99_check::result () const`

Return the status of the emptiness-check.

When [check\(\)](#) succeed, the status should be passed along to `spot::counter_example`.

This status should not be deleted, it is a pointer to a member of this class that will be deleted when the `couvreur99` object is deleted.

**11.16.3.12** `void spot::ec_statistics::set_states (unsigned n)` [inline, inherited]

**11.16.3.13** `int spot::ec_statistics::states () const` [inline, inherited]

**11.16.3.14** `int spot::ec_statistics::transitions () const` [inline, inherited]

## 11.16.4 Member Data Documentation

**11.16.4.1** `const tgba* spot::emptiness_check::a_` [protected, inherited]

The automaton.

**11.16.4.2** `couvreur99\_check\_status* spot::couvreur99_check::ecs_` [protected]

### 11.16.4.3 bool spot::couvreur99\_check::poprem\_ [protected]

Whether to store the state to be removed.

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

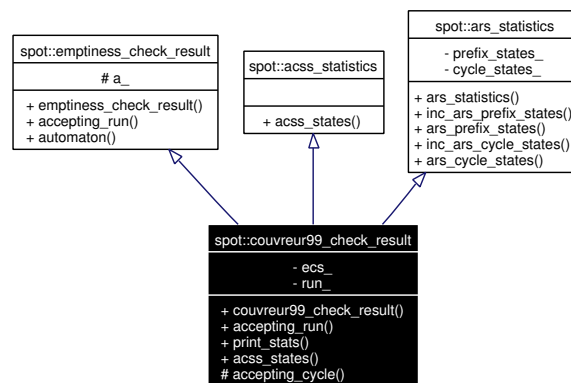
- [tgbaalgos/gtec/gtec.hh](#)

## 11.17 spot::couvreur99\_check\_result Class Reference

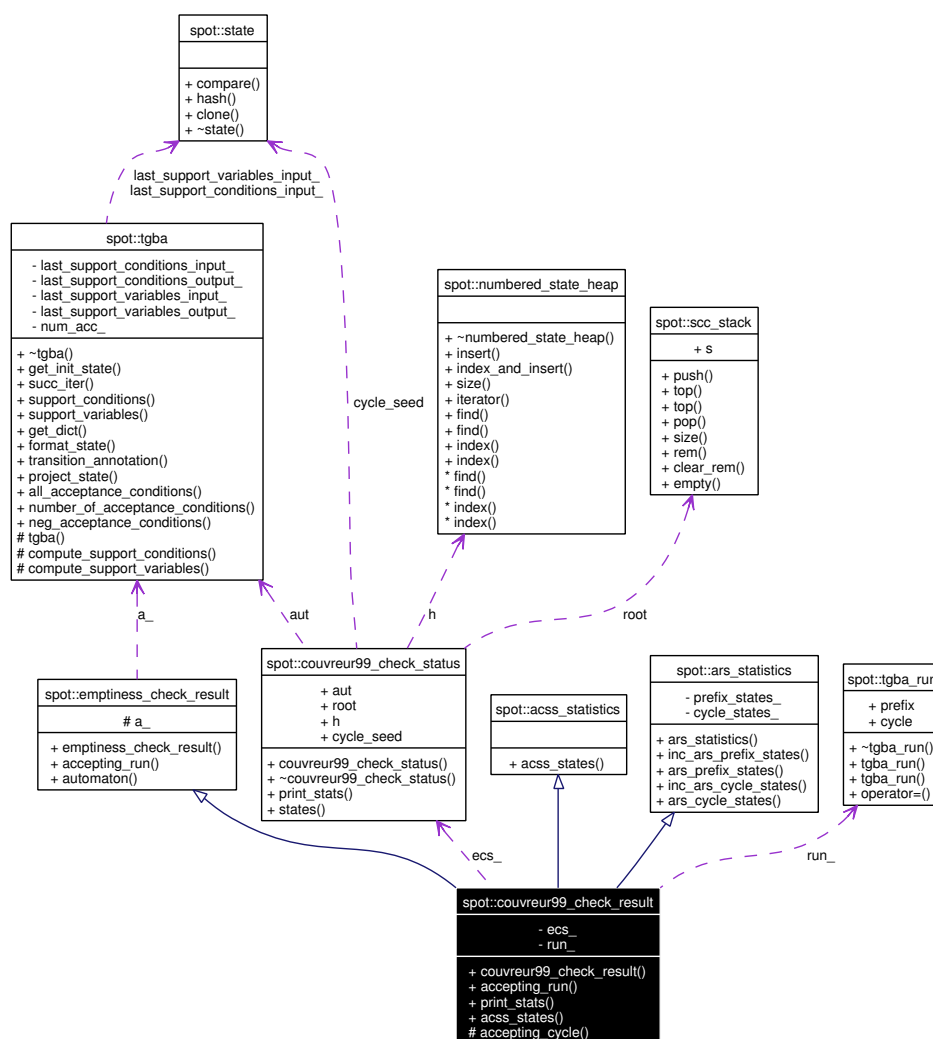
Compute a counter example from a [spot::couvreur99\\_check\\_status](#).

```
#include <tgbaalgos/gtec/ce.hh>
```

Inheritance diagram for spot::couvreur99\_check\_result:



Collaboration diagram for spot::couvreur99\_check\_result:



## Public Member Functions

- `couvereur99_check_result` (const `couvereur99_check_status` \*ecs)
- virtual `tgba` \* `accepting_run` ()  
*Return a run accepted by the automata passed to the emptiness check.*
- void `print_stats` (std::ostream &os) const
- virtual int `acss_states` () const  
*Number of states in the search space for the accepting cycle.*
- const `tgba` \* `automaton` () const  
*The automaton on which an `accepting_run()` was found.*
- void `inc_ars_prefix_states` ()
- int `ars_prefix_states` () const
- void `inc_ars_cycle_states` ()
- int `ars_cycle_states` () const

**Protected Member Functions**

- void `accepting_cycle` ()

**Protected Attributes**

- const `tgba` \* `a_`  
*The automaton.*

**Private Attributes**

- const `couvereur99_check_status` \* `ecs_`
- `tgba_run` \* `run_`

**11.17.1 Detailed Description**

Compute a counter example from a `spot::couvereur99_check_status`.

**11.17.2 Constructor & Destructor Documentation**

**11.17.2.1** `spot::couvereur99_check_result::couvereur99_check_result (const couvereur99_check_status * ecs)`

**11.17.3 Member Function Documentation**

**11.17.3.1** `void spot::couvereur99_check_result::accepting_cycle ()` [protected]

Called by `accepting_run()` to find a cycle which traverses all acceptance conditions in the accepted SCC.

**11.17.3.2** `virtual tgba_run* spot::couvereur99_check_result::accepting_run ()` [virtual]

Return a run accepted by the automata passed to the emptiness check.

This method might actually compute the acceptance run. (Not all emptiness check algorithms actually produce a counter-example as a side-effect of checking emptiness, some need some post-processing.)

This can also return 0 if the emptiness check algorithm cannot produce a counter example (that does not mean there is no counter-example; the mere existence of an instance of this class asserts the existence of a counter-example).

Reimplemented from `spot::emptiness_check_result`.

**11.17.3.3** `virtual int spot::couvereur99_check_result::acss_states () const` [virtual]

Number of states in the search space for the accepting cycle.

Implements `spot::acss_statistics`.

**11.17.3.4** `int spot::ars_statistics::ars_cycle_states () const` [inline, inherited]

**11.17.3.5** `int spot::ars_statistics::ars_prefix_states () const` [inline, inherited]

**11.17.3.6** `const tgba* spot::emptiness_check_result::automaton () const` [inline, inherited]

The automaton on which an [accepting\\_run\(\)](#) was found.

**11.17.3.7** `void spot::ars_statistics::inc_ars_cycle_states ()` [inline, inherited]

**11.17.3.8** `void spot::ars_statistics::inc_ars_prefix_states ()` [inline, inherited]

**11.17.3.9** `void spot::couvreur99_check_result::print_stats (std::ostream & os) const`

#### 11.17.4 Member Data Documentation

**11.17.4.1** `const tgba* spot::emptiness_check_result::a_` [protected, inherited]

The automaton.

**11.17.4.2** `const couvreur99_check_status* spot::couvreur99_check_result::ecs_` [private]

**11.17.4.3** `tgba_run* spot::couvreur99_check_result::run_` [private]

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

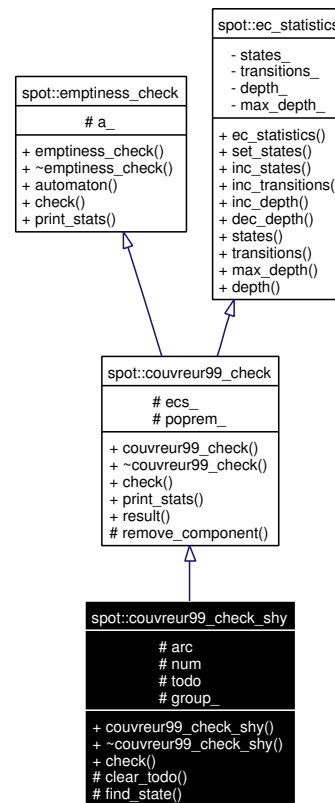
- [tgbaalgos/gtec/ce.hh](#)

## 11.18 spot::couvreur99\_check\_shy Class Reference

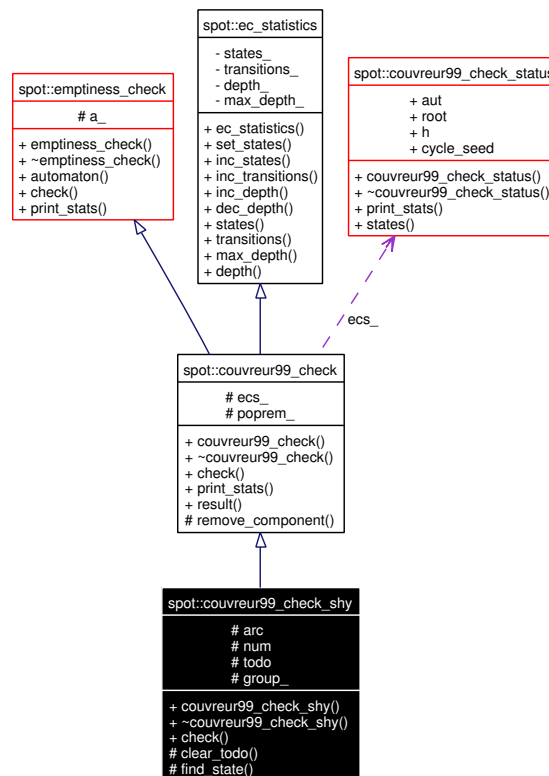
A version of [spot::couvreur99\\_check](#) that tries to visit known states first.

```
#include <tgbaalgos/gtec/gtec.hh>
```

Inheritance diagram for `spot::couvreur99_check_shy`:



Collaboration diagram for `spot::couvreur99_check_shy`:



## Public Member Functions

- `couvreur99_check_shy` (const `tgba` \*a, bool poprem=true, bool group=true, const `numbered_state_heap_factory` \*nshf=numbered\_state\_heap\_hash\_map\_factory::instance())
- virtual `~couvreur99_check_shy` ()
- virtual `emptiness_check_result` \* `check` ()  
*Check whether the automaton's language is empty.*
- virtual `std::ostream` & `print_stats` (std::ostream &os) const  
*Print statistics, if any.*
- const `couvreur99_check_status` \* `result` () const  
*Return the status of the emptiness-check.*
- const `tgba` \* `automaton` () const  
*The automaton that this emptiness-check inspects.*
- void `set_states` (unsigned n)
- void `inc_states` ()
- void `inc_transitions` ()
- void `inc_depth` (unsigned n=1)
- void `dec_depth` (unsigned n=1)
- int `states` () const
- int `transitions` () const

- `int max_depth () const`
- `int depth () const`

### Protected Types

- `typedef std::list< successor > succ_queue`
- `typedef std::list< todo_item > todo_list`

### Protected Member Functions

- `void clear_todo ()`
- `virtual int * find_state (const state *s)`  
*find the SCC number of a unprocessed state.*
- `void remove_component (const state *start_delete)`  
*Remove a strongly component from the hash.*

### Protected Attributes

- `std::stack< bdd > arc`
- `int num`
- `todo_list todo`
- `bool group_`
- `couvreur99_check_status * ecs_`
- `bool poprem_`  
*Whether to store the state to be removed.*
- `const tgba * a_`  
*The automaton.*

### Classes

- `struct successor`
- `struct todo_item`

#### 11.18.1 Detailed Description

A version of `spot::couvreur99_check` that tries to visit known states first.

If *group* is true (the default), the successors of all the states that belong to the same SCC will be considered when choosing a successor. Otherwise, only the successor of the topmost state on the DFS stack are considered.

See the documentation for `spot::couvreur99_check`

#### 11.18.2 Member Typedef Documentation

##### 11.18.2.1 `typedef std::list<successor> spot::couvreur99_check_shy::succ_queue` [protected]



11.18.2.2 `typedef std::list<todo_item> spot::couvreur99_check_shy::todo_list` [protected]

### 11.18.3 Constructor & Destructor Documentation

11.18.3.1 `spot::couvreur99_check_shy::couvreur99_check_shy (const tgba * a, bool poprem = true, bool group = true, const numbered_state_heap_factory * nshf = numbered_state_heap_hash_map_factory::instance())`

11.18.3.2 `virtual spot::couvreur99_check_shy::~couvreur99_check_shy ()` [virtual]

### 11.18.4 Member Function Documentation

11.18.4.1 `const tgba* spot::emptiness_check::automaton () const` [inline, inherited]

The automaton that this emptiness-check inspects.

11.18.4.2 `virtual emptiness_check_result* spot::couvreur99_check_shy::check ()` [virtual]

Check whether the automaton's language is empty.

Reimplemented from `spot::couvreur99_check`.

11.18.4.3 `void spot::couvreur99_check_shy::clear_todo ()` [protected]

11.18.4.4 `void spot::ec_statistics::dec_depth (unsigned n = 1)` [inline, inherited]

11.18.4.5 `int spot::ec_statistics::depth () const` [inline, inherited]

11.18.4.6 `virtual int* spot::couvreur99_check_shy::find_state (const state * s)` [protected, virtual]

find the SCC number of a unprocessed state.

Sometimes we want to modify some of the above structures when looking up a new state. This happens for instance when `find()` must perform inclusion checking and add new states to process to TODO during this step. (Because TODO must be known, sub-classing `spot::numbered_state_heap` is not enough.) Then overriding this method is the way to go.

11.18.4.7 `void spot::ec_statistics::inc_depth (unsigned n = 1)` [inline, inherited]

11.18.4.8 `void spot::ec_statistics::inc_states ()` [inline, inherited]

11.18.4.9 `void spot::ec_statistics::inc_transitions ()` [inline, inherited]

11.18.4.10 `int spot::ec_statistics::max_depth () const` [inline, inherited]

**11.18.4.11** `virtual std::ostream& spot::couvreur99_check::print_stats (std::ostream & os) const` [virtual, inherited]

Print statistics, if any.

Reimplemented from [spot::emptiness\\_check](#).

**11.18.4.12** `void spot::couvreur99_check::remove_component (const state * start_delete)` [protected, inherited]

Remove a strongly component from the hash.

This function remove all accessible state from a given state. In other words, it removes the strongly connected component that contains this state.

**11.18.4.13** `const couvreur99\_check\_status* spot::couvreur99_check::result () const` [inherited]

Return the status of the emptiness-check.

When [check\(\)](#) succeed, the status should be passed along to `spot::counter_example`.

This status should not be deleted, it is a pointer to a member of this class that will be deleted when the `couvreur99` object is deleted.

**11.18.4.14** `void spot::ec_statistics::set_states (unsigned n)` [inline, inherited]

**11.18.4.15** `int spot::ec_statistics::states () const` [inline, inherited]

**11.18.4.16** `int spot::ec_statistics::transitions () const` [inline, inherited]

## 11.18.5 Member Data Documentation

**11.18.5.1** `const tgba* spot::emptiness_check::a_` [protected, inherited]

The automaton.

**11.18.5.2** `std::stack<bdd> spot::couvreur99_check_shy::arc` [protected]

**11.18.5.3** `couvreur99\_check\_status* spot::couvreur99_check::ecs_` [protected, inherited]

**11.18.5.4** `bool spot::couvreur99_check_shy::group_` [protected]

**11.18.5.5** `int spot::couvreur99_check_shy::num` [protected]

**11.18.5.6** `bool spot::couvreur99_check::poprem_` [protected, inherited]

Whether to store the state to be removed.

11.18.5.7 `todo_list spot::couvreur99_check_shy::todo` [protected]

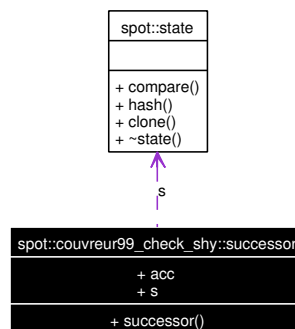
The documentation for this class was generated from the following file:

- `tgbaalgos/gtec/gtec.hh`

11.19 `spot::couvreur99_check_shy::successor` Struct Reference

```
#include <tgbaalgos/gtec/gtec.hh>
```

Collaboration diagram for `spot::couvreur99_check_shy::successor`:



## Public Member Functions

- `successor` (bdd `acc`, const `spot::state *s`)

## Public Attributes

- bdd `acc`
- const `spot::state *s`

## 11.19.1 Constructor &amp; Destructor Documentation

11.19.1.1 `spot::couvreur99_check_shy::successor::successor` (bdd `acc`, const `spot::state *s`)  
[inline]

## 11.19.2 Member Data Documentation

11.19.2.1 bdd `spot::couvreur99_check_shy::successor::acc`

11.19.2.2 const `spot::state*` `spot::couvreur99_check_shy::successor::s`

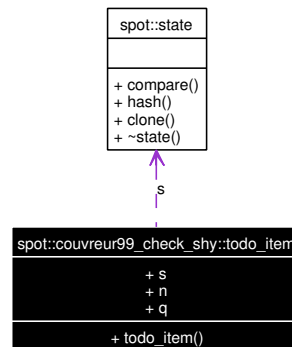
The documentation for this struct was generated from the following file:

- `tgbaalgos/gtec/gtec.hh`

## 11.20 spot::couvreur99\_check\_shy::todo\_item Struct Reference

```
#include <tgbaalgos/gtec/gtec.hh>
```

Collaboration diagram for spot::couvreur99\_check\_shy::todo\_item:



### Public Member Functions

- [todo\\_item](#) (const [state](#) \*[s](#), int [n](#))

### Public Attributes

- const [state](#) \* [s](#)
- int [n](#)
- [succ\\_queue](#) [q](#)

### 11.20.1 Constructor & Destructor Documentation

11.20.1.1 [spot::couvreur99\\_check\\_shy::todo\\_item::todo\\_item](#) (const [state](#) \* [s](#), int [n](#)) `[inline]`

### 11.20.2 Member Data Documentation

11.20.2.1 int [spot::couvreur99\\_check\\_shy::todo\\_item::n](#)

11.20.2.2 [succ\\_queue](#) [spot::couvreur99\\_check\\_shy::todo\\_item::q](#)

11.20.2.3 const [state](#)\* [spot::couvreur99\\_check\\_shy::todo\\_item::s](#)

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

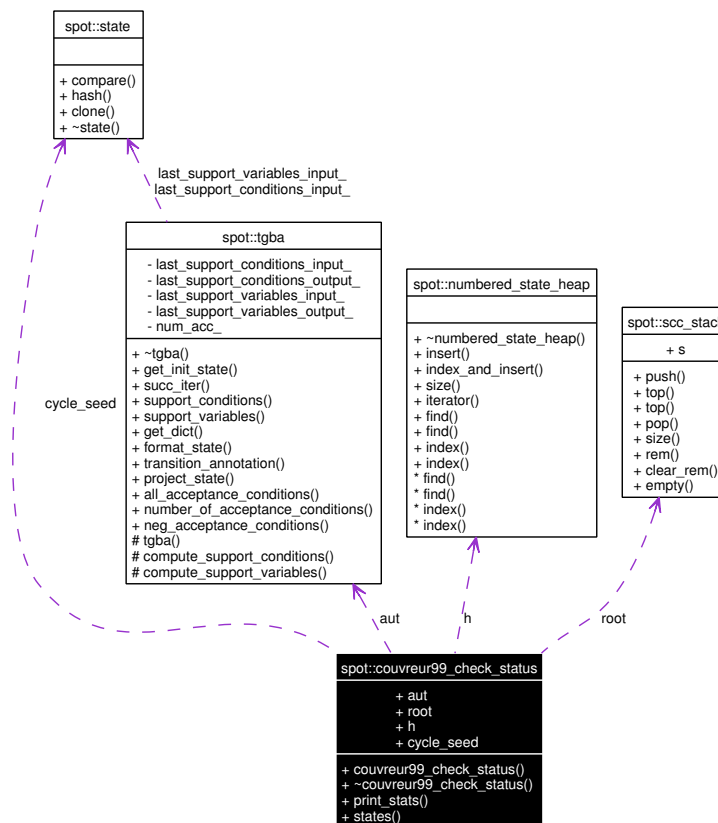
- [tgbaalgos/gtec/gtec.hh](#)

## 11.21 spot::couvreur99\_check\_status Class Reference

The status of the emptiness-check on success.

```
#include <tgbaalgos/gtec/status.hh>
```

Collaboration diagram for spot::couvreur99\_check\_status:



### Public Member Functions

- `couvreur99_check_status` (const `tgba` \*`aut`, const `numbered_state_heap_factory` \*`nshf`)
- `~couvreur99_check_status` ()
- void `print_stats` (std::ostream &`os`) const  
*Output statistics about this object.*
- int `states` () const  
*Return the number of states visited by the search.*

### Public Attributes

- const `tgba` \* `aut`
- `scc_stack` `root`
- `numbered_state_heap` \* `h`

*Heap of visited states.*

- const [state](#) \* [cycle\\_seed](#)

### 11.21.1 Detailed Description

The status of the emptiness-check on success.

This contains everything needed to construct a counter-example: the automata, the stack of SCCs traversed by the counter-example, and the heap of visited states with their indexes.

### 11.21.2 Constructor & Destructor Documentation

**11.21.2.1** `spot::couvreur99_check_status::couvreur99_check_status (const tgba * aut, const numbered\_state\_heap\_factory * nshf)`

**11.21.2.2** `spot::couvreur99_check_status::~~couvreur99_check_status ()`

### 11.21.3 Member Function Documentation

**11.21.3.1** `void spot::couvreur99_check_status::print_stats (std::ostream & os) const`

Output statistics about this object.

**11.21.3.2** `int spot::couvreur99_check_status::states () const`

Return the number of states visited by the search.

### 11.21.4 Member Data Documentation

**11.21.4.1** `const tgba* spot::couvreur99_check_status::aut`

**11.21.4.2** `const state* spot::couvreur99_check_status::cycle_seed`

**11.21.4.3** `numbered\_state\_heap* spot::couvreur99_check_status::h`

Heap of visited states.

**11.21.4.4** `scc\_stack spot::couvreur99_check_status::root`

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

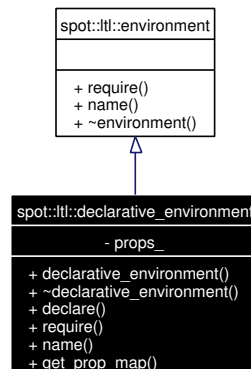
- [tgbaalgos/gtec/status.hh](#)

## 11.22 spot::ltl::declarative\_environment Class Reference

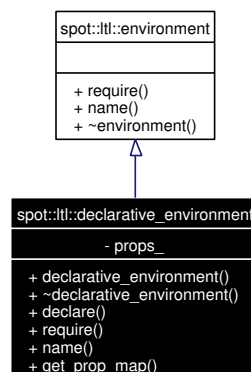
A declarative environment.

```
#include <ltlenv/declenv.hh>
```

Inheritance diagram for spot::ltl::declarative\_environment:



Collaboration diagram for spot::ltl::declarative\_environment:



## Public Types

- typedef `std::map< const std::string, ltl::atomic\_prop * >` `prop_map`

## Public Member Functions

- [declarative\\_environment](#) ()
- [~declarative\\_environment](#) ()
- bool [declare](#) (const std::string &prop\_str)
- virtual [ltl::formula](#) \* [require](#) (const std::string &prop\_str)  
*Obtain the formula associated to prop\_str.*
- virtual const std::string & [name](#) ()  
*Get the name of the environment.*
- const [prop\\_map](#) & [get\\_prop\\_map](#) () const  
*Get the map of atomic proposition known to this environment.*

**Private Attributes**

- `prop_map` `props_`

**11.22.1 Detailed Description**

A declarative environment.

This environment recognizes all atomic propositions that have been previously declared. It will reject other.

**11.22.2 Member Typedef Documentation**

**11.22.2.1** `typedef std::map<const std::string, ltl::atomic_prop*> spot::ltl::declarative_environment::prop_map`

**11.22.3 Constructor & Destructor Documentation**

**11.22.3.1** `spot::ltl::declarative_environment::declarative_environment ()`

**11.22.3.2** `spot::ltl::declarative_environment::~~declarative_environment ()`

**11.22.4 Member Function Documentation**

**11.22.4.1** `bool spot::ltl::declarative_environment::declare (const std::string & prop_str)`

Declare an atomic proposition. Return false iff the proposition was already declared.

**11.22.4.2** `const prop_map& spot::ltl::declarative_environment::get_prop_map () const`

Get the map of atomic proposition known to this environment.

**11.22.4.3** `virtual const std::string& spot::ltl::declarative_environment::name () [virtual]`

Get the name of the environment.

Implements `spot::ltl::environment`.

**11.22.4.4** `virtual ltl::formula* spot::ltl::declarative_environment::require (const std::string & prop_str) [virtual]`

Obtain the formula associated to *prop\_str*.

Usually *prop\_str*, is the name of an atomic proposition, and `spot::ltl::require` simply returns the associated `spot::ltl::atomic_prop`.

Note this is not a `const` method. Some environments will "create" the atomic proposition when requested.

We return a `spot::ltl::formula` instead of an `spot::ltl::atomic_prop`, because this will allow nifty tricks (e.g., we could name formulae in an environment, and let the parser build a larger tree from these).

**Returns:**

0 iff *prop\_str* is not part of the environment, or the associated `spot::ltl::formula` otherwise.

Implements `spot::ltl::environment`.



### 11.22.5 Member Data Documentation

#### 11.22.5.1 `prop_map` `spot::ltl::declarative_environment::props_` [private]

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

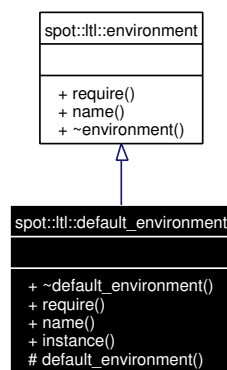
- `ltlenv/declenv.hh`

## 11.23 spot::ltl::default\_environment Class Reference

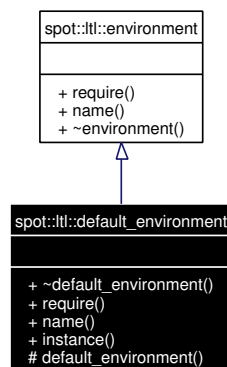
A laxist environment.

```
#include <ltlenv/defaultenv.hh>
```

Inheritance diagram for `spot::ltl::default_environment`:



Collaboration diagram for `spot::ltl::default_environment`:



### Public Member Functions

- virtual `~default_environment` ()
- virtual `formula * require` (const std::string &prop\_str)

*Obtain the formula associated to prop\_str.*

- virtual const std::string & `name` ()  
*Get the name of the environment.*

### Static Public Member Functions

- static `default_environment` & `instance` ()  
*Get the sole instance of `spot::ltl::default_environment`.*

### Protected Member Functions

- `default_environment` ()

#### 11.23.1 Detailed Description

A laxist environment.

This environment recognizes all atomic propositions.

This is a singleton. Use `default_environment::instance()` to obtain the instance.

#### 11.23.2 Constructor & Destructor Documentation

**11.23.2.1** virtual `spot::ltl::default_environment::~~default_environment` () [virtual]

**11.23.2.2** `spot::ltl::default_environment::default_environment` () [protected]

#### 11.23.3 Member Function Documentation

**11.23.3.1** static `default_environment`& `spot::ltl::default_environment::instance` () [static]

Get the sole instance of `spot::ltl::default_environment`.

**11.23.3.2** virtual const std::string& `spot::ltl::default_environment::name` () [virtual]

Get the name of the environment.

Implements `spot::ltl::environment`.

**11.23.3.3** virtual `formula`\* `spot::ltl::default_environment::require` (const std::string & *prop\_str*) [virtual]

Obtain the formula associated to *prop\_str*.

Usually *prop\_str*, is the name of an atomic proposition, and `spot::ltl::require` simply returns the associated `spot::ltl::atomic_prop`.

Note this is not a `const` method. Some environments will "create" the atomic proposition when requested.

We return a `spot::ltl::formula` instead of an `spot::ltl::atomic_prop`, because this will allow nifty tricks (e.g., we could name formulae in an environment, and let the parser build a larger tree from these).

**Returns:**

0 iff *prop\_str* is not part of the environment, or the associated [spot::ltl::formula](#) otherwise.

Implements [spot::ltl::environment](#).

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

- [ltlenv/defaultenv.hh](#)

**11.24 spot::delayed\_simulation\_relation Class Reference**

```
#include <tgba/tgbareduc.hh>
```

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

- [tgba/tgbareduc.hh](#)

**11.25 spot::direct\_simulation\_relation Class Reference**

```
#include <tgba/tgbareduc.hh>
```

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

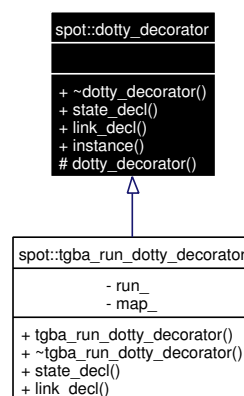
- [tgba/tgbareduc.hh](#)

**11.26 spot::dotty\_decorator Class Reference**

Choose state and link styles for [spot::dotty\\_reachable](#).

```
#include <tgbaalgos/dottydec.hh>
```

Inheritance diagram for `spot::dotty_decorator`:

**Public Member Functions**

- virtual `~dotty_decorator()`
- virtual `std::string state_decl` (const `tgba` \*a, const `state` \*s, int n, `tgba_succ_iterator` \*si, const `std::string` &label)

*Compute the style of a state.*

- virtual std::string link\_decl (const tgba \*a, const state \*in\_s, int in, const state \*out\_s, int out, const tgba\_succ\_iterator \*si, const std::string &label)

*Compute the style of a link.*

### Static Public Member Functions

- static dotty\_decorator \* instance ()

*Get the unique instance of the default dotty\_decorator.*

### Protected Member Functions

- dotty\_decorator ()

#### 11.26.1 Detailed Description

Choose state and link styles for spot::dotty\_reachable.

#### 11.26.2 Constructor & Destructor Documentation

**11.26.2.1** virtual spot::dotty\_decorator::~~dotty\_decorator () [virtual]

**11.26.2.2** spot::dotty\_decorator::dotty\_decorator () [protected]

#### 11.26.3 Member Function Documentation

**11.26.3.1** static dotty\_decorator\* spot::dotty\_decorator::instance () [static]

Get the unique instance of the default dotty\_decorator.

**11.26.3.2** virtual std::string spot::dotty\_decorator::link\_decl (const tgba \* a, const state \* in\_s, int in, const state \* out\_s, int out, const tgba\_succ\_iterator \* si, const std::string & label) [virtual]

Compute the style of a link.

This function should output a string of the form [label="foo", style=bar, ...]. The default implementation will simply output [label="LABEL" ] with LABEL replaced by the value of *label*.

#### Parameters:

*a* the automaton being drawn

*in\_s* the source state of the transition being drawn (owned by the caller)

*in* the unique number associated to *in\_s*

*out\_s* the destination state of the transition being drawn (owned by the caller)

*out* the unique number associated to *out\_s*

*si* an iterator over the successors of *in\_s*, pointing to the current transition (owned by the caller and cannot be iterated)

*label* the computed name of this state

Reimplemented in `spot::tgba_run_dotty_decorator`.

**11.26.3.3** `virtual std::string spot::dotty_decorator::state_decl (const tgba * a, const state * s, int n, tgba_succ_iterator * si, const std::string & label) [virtual]`

Compute the style of a state.

This function should output a string of the form `[label="foo", style=bar, ...]`. The default implementation will simply output `[label="LABEL"]` with LABEL replaced by the value of *label*.

#### Parameters:

*a* the automaton being drawn

*s* the state being drawn (owned by the caller)

*n* a unique number for this state

*si* an iterator over the successors of this state (owned by the caller, but can be freely iterated)

*label* the computed name of this state

Reimplemented in `spot::tgba_run_dotty_decorator`.

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

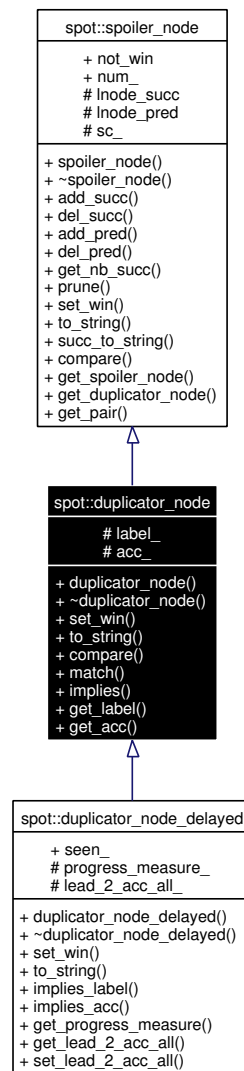
- `tgbaalgos/dottydec.hh`

## 11.27 `spot::duplicator_node` Class Reference

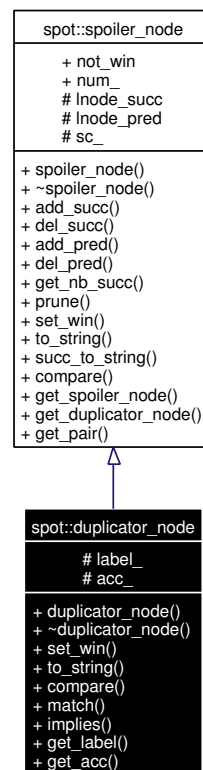
Duplicator node of parity game graph.

```
#include <tgbaalgos/reductgba_sim.hh>
```

Inheritance diagram for `spot::duplicator_node`:



Collaboration diagram for spot::duplicator\_node:



## Public Member Functions

- `duplicator_node` (const `state` \*d\_node, const `state` \*s\_node, bdd l, bdd a, int num)
- virtual `~duplicator_node` ()
- virtual bool `set_win` ()
- virtual std::string `to_string` (const `tgba` \*a)
- virtual bool `compare` (`spoiler_node` \*n)
- bool `match` (bdd l, bdd a)
- bool `implies` (bdd l, bdd a)
- bdd `get_label` () const
- bdd `get_acc` () const
- bool `add_succ` (`spoiler_node` \*n)

*Add a successor. Return true if n wasn't yet in the list of successor, false otherwise.*

- void `del_succ` (`spoiler_node` \*n)
- virtual void `add_pred` (`spoiler_node` \*n)
- virtual void `del_pred` ()
- int `get_nb_succ` ()
- bool `prune` ()
- virtual std::string `succ_to_string` ()
- const `state` \* `get_spoiler_node` ()
- const `state` \* `get_duplicator_node` ()
- `state_couple` \* `get_pair` ()

**Public Attributes**

- bool [not\\_win](#)
- int [num\\_](#)

**Protected Attributes**

- bdd [label\\_](#)
- bdd [acc\\_](#)
- [sn\\_v](#) \* [lnode\\_succ](#)
- [sn\\_v](#) \* [lnode\\_pred](#)
- [state\\_couple](#) \* [sc\\_](#)

**11.27.1 Detailed Description**

Duplicator node of parity game graph.

**11.27.2 Constructor & Destructor Documentation**

**11.27.2.1** `spot::duplicator_node::duplicator_node (const state * d_node, const state * s_node, bdd l, bdd a, int num)`

**11.27.2.2** `virtual spot::duplicator_node::~duplicator\_node ()` [virtual]

**11.27.3 Member Function Documentation**

**11.27.3.1** `virtual void spot::spoiler_node::add_pred (spoiler\_node * n)` [virtual, inherited]

**11.27.3.2** `bool spot::spoiler_node::add_succ (spoiler\_node * n)` [inherited]

Add a successor. Return true if *n* wasn't yet in the list of successor, false otherwise.

**11.27.3.3** `virtual bool spot::duplicator_node::compare (spoiler\_node * n)` [virtual]

Reimplemented from [spot::spoiler\\_node](#).

**11.27.3.4** `virtual void spot::spoiler_node::del_pred ()` [virtual, inherited]

**11.27.3.5** `void spot::spoiler_node::del_succ (spoiler\_node * n)` [inherited]

**11.27.3.6** `bdd spot::duplicator_node::get_acc () const`

**11.27.3.7** `const state* spot::spoiler_node::get_duplicator_node ()` [inherited]

**11.27.3.8** `bdd spot::duplicator_node::get_label () const`



11.27.3.9 `int spot::spoiler_node::get_nb_succ ()` [inherited]

11.27.3.10 `state_couple* spot::spoiler_node::get_pair ()` [inherited]

11.27.3.11 `const state* spot::spoiler_node::get_spoiler_node ()` [inherited]

11.27.3.12 `bool spot::duplicator_node::implies (bdd l, bdd a)`

11.27.3.13 `bool spot::duplicator_node::match (bdd l, bdd a)`

11.27.3.14 `bool spot::spoiler_node::prune ()` [inherited]

11.27.3.15 `virtual bool spot::duplicator_node::set_win ()` [virtual]

Reimplemented from `spot::spoiler_node`.

Reimplemented in `spot::duplicator_node_delayed`.

11.27.3.16 `virtual std::string spot::spoiler_node::succ_to_string ()` [virtual, inherited]

11.27.3.17 `virtual std::string spot::duplicator_node::to_string (const tgba * a)` [virtual]

Reimplemented from `spot::spoiler_node`.

Reimplemented in `spot::duplicator_node_delayed`.

## 11.27.4 Member Data Documentation

11.27.4.1 `bdd spot::duplicator_node::acc_` [protected]

11.27.4.2 `bdd spot::duplicator_node::label_` [protected]

11.27.4.3 `sn_v* spot::spoiler_node::lnode_pred` [protected, inherited]

11.27.4.4 `sn_v* spot::spoiler_node::lnode_succ` [protected, inherited]

11.27.4.5 `bool spot::spoiler_node::not_win` [inherited]

11.27.4.6 `int spot::spoiler_node::num_` [inherited]

11.27.4.7 `state_couple* spot::spoiler_node::sc_` [protected, inherited]

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

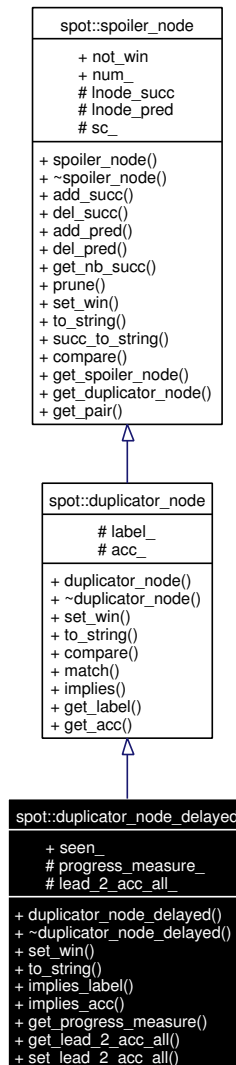
- `tgbaalgos/reductgba_sim.hh`

## 11.28 spot::duplicator\_node\_delayed Class Reference

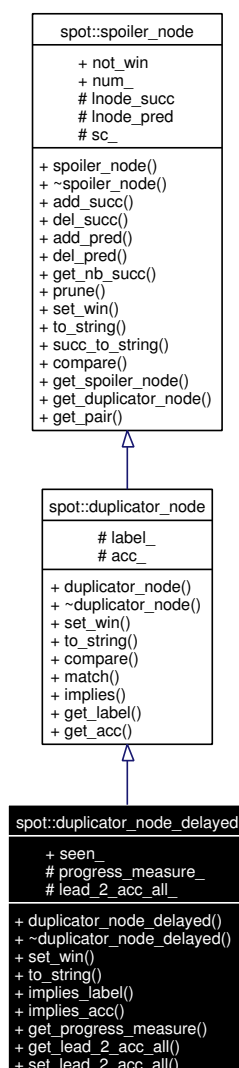
Duplicator node of parity game graph for delayed simulation.

```
#include <tgbaalgos/reductgba_sim.hh>
```

Inheritance diagram for spot::duplicator\_node\_delayed:



Collaboration diagram for spot::duplicator\_node\_delayed:



## Public Member Functions

- `duplicator_node_delayed` (const `state` \*`d_node`, const `state` \*`s_node`, bdd `l`, bdd `a`, int `num`)
- `~duplicator_node_delayed` ()
- bool `set_win` ()

*Return true if the progress\_measure has changed.*

- virtual std::string `to_string` (const `tgba` \*`a`)
- bool `implies_label` (bdd `l`)
- bool `implies_acc` (bdd `a`)
- int `get_progress_measure` ()
- bool `get_lead_2_acc_all` ()
- bool `set_lead_2_acc_all` (bdd `acc`=bddfalse)
- virtual bool `compare` (`spoiler_node` \*`n`)
- bool `match` (bdd `l`, bdd `a`)
- bool `implies` (bdd `l`, bdd `a`)

- bdd [get\\_label](#) () const
- bdd [get\\_acc](#) () const
- bool [add\\_succ](#) (spoiler\_node \*n)

*Add a successor. Return true if n wasn't yet in the list of successor, false otherwise.*

- void [del\\_succ](#) (spoiler\_node \*n)
- virtual void [add\\_pred](#) (spoiler\_node \*n)
- virtual void [del\\_pred](#) ()
- int [get\\_nb\\_succ](#) ()
- bool [prune](#) ()
- virtual std::string [succ\\_to\\_string](#) ()
- const state \* [get\\_spoiler\\_node](#) ()
- const state \* [get\\_duplicator\\_node](#) ()
- state\_couple \* [get\\_pair](#) ()

### Public Attributes

- bool [seen\\_](#)
- bool [not\\_win](#)
- int [num\\_](#)

### Protected Attributes

- int [progress\\_measure\\_](#)
- bool [lead\\_2\\_acc\\_all\\_](#)
- bdd [label\\_](#)
- bdd [acc\\_](#)
- sn\_v \* [lnode\\_succ](#)
- sn\_v \* [lnode\\_pred](#)
- state\_couple \* [sc\\_](#)

## 11.28.1 Detailed Description

Duplicator node of parity game graph for delayed simulation.

## 11.28.2 Constructor & Destructor Documentation

**11.28.2.1** spot::duplicator\_node\_delayed::duplicator\_node\_delayed (const state \* [d\\_node](#), const state \* [s\\_node](#), bdd [l](#), bdd [a](#), int [num](#))

**11.28.2.2** spot::duplicator\_node\_delayed::~~duplicator\_node\_delayed ()

## 11.28.3 Member Function Documentation

**11.28.3.1** virtual void spot::spoiler\_node::add\_pred (spoiler\_node \* [n](#)) [virtual, inherited]

**11.28.3.2** `bool spot::spoiler_node::add_succ (spoiler_node * n)` [inherited]

Add a successor. Return true if  $n$  wasn't yet in the list of successor, false otherwise.

**11.28.3.3** `virtual bool spot::duplicator_node::compare (spoiler_node * n)` [virtual, inherited]

Reimplemented from `spot::spoiler_node`.

**11.28.3.4** `virtual void spot::spoiler_node::del_pred ()` [virtual, inherited]

**11.28.3.5** `void spot::spoiler_node::del_succ (spoiler_node * n)` [inherited]

**11.28.3.6** `bdd spot::duplicator_node::get_acc () const` [inherited]

**11.28.3.7** `const state* spot::spoiler_node::get_duplicator_node ()` [inherited]

**11.28.3.8** `bdd spot::duplicator_node::get_label () const` [inherited]

**11.28.3.9** `bool spot::duplicator_node_delayed::get_lead_2_acc_all ()`

**11.28.3.10** `int spot::spoiler_node::get_nb_succ ()` [inherited]

**11.28.3.11** `state_couple* spot::spoiler_node::get_pair ()` [inherited]

**11.28.3.12** `int spot::duplicator_node_delayed::get_progress_measure ()`

**11.28.3.13** `const state* spot::spoiler_node::get_spoiler_node ()` [inherited]

**11.28.3.14** `bool spot::duplicator_node::implies (bdd l, bdd a)` [inherited]

**11.28.3.15** `bool spot::duplicator_node_delayed::implies_acc (bdd a)`

**11.28.3.16** `bool spot::duplicator_node_delayed::implies_label (bdd l)`

**11.28.3.17** `bool spot::duplicator_node::match (bdd l, bdd a)` [inherited]

**11.28.3.18** `bool spot::spoiler_node::prune ()` [inherited]

**11.28.3.19** `bool spot::duplicator_node_delayed::set_lead_2_acc_all (bdd acc = bddfalse)`

**11.28.3.20** `bool spot::duplicator_node_delayed::set_win ()` [virtual]

Return true if the progress\_measure has changed.

Reimplemented from [spot::duplicator\\_node](#).

**11.28.3.21** `virtual std::string spot::spoiler_node::succ_to_string ()` [virtual, inherited]**11.28.3.22** `virtual std::string spot::duplicator_node_delayed::to_string (const tgba * a)` [virtual]

Reimplemented from [spot::duplicator\\_node](#).

**11.28.4 Member Data Documentation****11.28.4.1** `bdd spot::duplicator\_node::acc\_` [protected, inherited]**11.28.4.2** `bdd spot::duplicator\_node::label\_` [protected, inherited]**11.28.4.3** `bool spot::duplicator\_node\_delayed::lead\_2\_acc\_all\_` [protected]**11.28.4.4** `sn_v* spot::spoiler\_node::lnode\_pred` [protected, inherited]**11.28.4.5** `sn_v* spot::spoiler\_node::lnode\_succ` [protected, inherited]**11.28.4.6** `bool spot::spoiler\_node::not\_win` [inherited]**11.28.4.7** `int spot::spoiler\_node::num\_` [inherited]**11.28.4.8** `int spot::duplicator\_node\_delayed::progress\_measure\_` [protected]**11.28.4.9** `state_couple* spot::spoiler\_node::sc\_` [protected, inherited]**11.28.4.10** `bool spot::duplicator\_node\_delayed::seen\_`

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

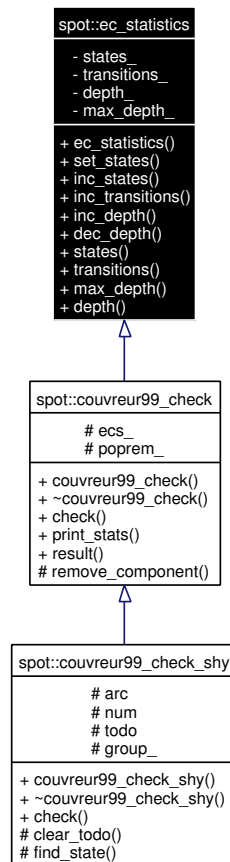
- [tgbaalgos/reductgba\\_sim.hh](#)

**11.29 spot::ec\_statistics Class Reference**

Emptiness-check statistics.

```
#include <tgbaalgos/emptiness_stats.hh>
```

Inheritance diagram for `spot::ec_statistics`:



## Public Member Functions

- [ec\\_statistics](#) ()
- void [set\\_states](#) (unsigned n)
- void [inc\\_states](#) ()
- void [inc\\_transitions](#) ()
- void [inc\\_depth](#) (unsigned n=1)
- void [dec\\_depth](#) (unsigned n=1)
- int [states](#) () const
- int [transitions](#) () const
- int [max\\_depth](#) () const
- int [depth](#) () const

## Private Attributes

- unsigned [states\\_](#)
- unsigned [transitions\\_](#)  
*number of distinct visited states*
- unsigned [depth\\_](#)  
*number of visited transitions*

- unsigned [max\\_depth\\_](#)  
*maximal depth of the stack(s)*

### 11.29.1 Detailed Description

Emptiness-check statistics.

Implementations of [spot::emptiness\\_check](#) may also implement this interface. Try to dynamic\_cast the [spot::emptiness\\_check](#) pointer to know whether these statistics are available.

### 11.29.2 Constructor & Destructor Documentation

**11.29.2.1** `spot::ec_statistics::ec_statistics ()` [inline]

### 11.29.3 Member Function Documentation

**11.29.3.1** `void spot::ec_statistics::dec_depth (unsigned n = 1)` [inline]

**11.29.3.2** `int spot::ec_statistics::depth () const` [inline]

**11.29.3.3** `void spot::ec_statistics::inc_depth (unsigned n = 1)` [inline]

**11.29.3.4** `void spot::ec_statistics::inc_states ()` [inline]

**11.29.3.5** `void spot::ec_statistics::inc_transitions ()` [inline]

**11.29.3.6** `int spot::ec_statistics::max_depth () const` [inline]

**11.29.3.7** `void spot::ec_statistics::set_states (unsigned n)` [inline]

**11.29.3.8** `int spot::ec_statistics::states () const` [inline]

**11.29.3.9** `int spot::ec_statistics::transitions () const` [inline]

### 11.29.4 Member Data Documentation

**11.29.4.1** unsigned [spot::ec\\_statistics::depth\\_](#) [private]  
number of visited transitions

**11.29.4.2** unsigned [spot::ec\\_statistics::max\\_depth\\_](#) [private]  
maximal depth of the stack(s)



11.29.4.3 unsigned [spot::ec\\_statistics::states\\_](#) [private]

11.29.4.4 unsigned [spot::ec\\_statistics::transitions\\_](#) [private]

number of distinct visited states

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

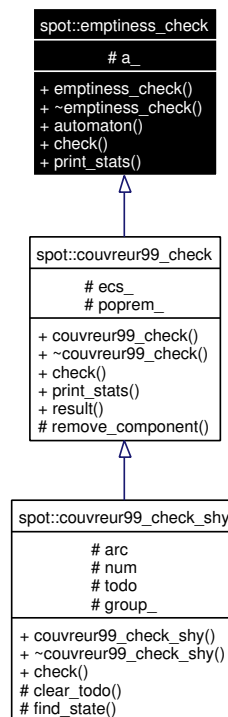
- [tgbaalgos/emptiness\\_stats.hh](#)

## 11.30 spot::emptiness\_check Class Reference

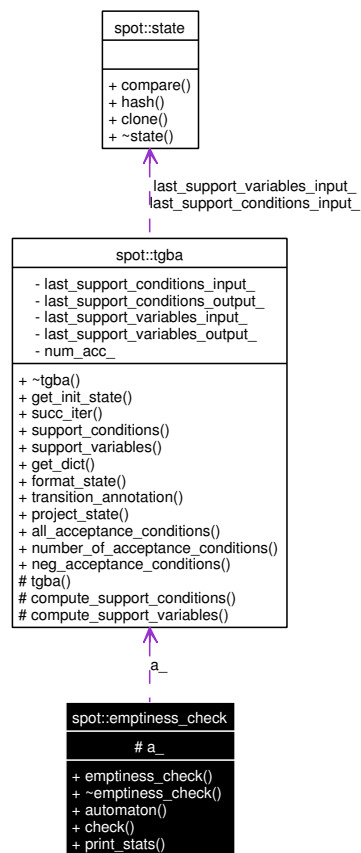
Common interface to emptiness check algorithms.

```
#include <tgbaalgos/emptiness.hh>
```

Inheritance diagram for spot::emptiness\_check:



Collaboration diagram for spot::emptiness\_check:



## Public Member Functions

- [emptiness\\_check](#) (const [tgba](#) \*a)
- virtual [~emptiness\\_check](#) ()
- const [tgba](#) \* [automaton](#) () const  
*The automaton that this emptiness-check inspects.*
- virtual [emptiness\\_check\\_result](#) \* [check](#) ()=0  
*Check whether the automaton contain an accepting run.*
- virtual std::ostream & [print\\_stats](#) (std::ostream &os) const  
*Print statistics, if any.*

## Protected Attributes

- const [tgba](#) \* [a\\_](#)  
*The automaton.*

### 11.30.1 Detailed Description

Common interface to emptiness check algorithms.

### 11.30.2 Constructor & Destructor Documentation

**11.30.2.1** `spot::emptiness_check::emptiness_check (const tgba * a)` `[inline]`

**11.30.2.2** `virtual spot::emptiness_check::~~emptiness_check ()` `[virtual]`

### 11.30.3 Member Function Documentation

**11.30.3.1** `const tgba* spot::emptiness_check::automaton () const` `[inline]`

The automaton that this emptiness-check inspects.

**11.30.3.2** `virtual emptiness\_check\_result* spot::emptiness_check::check ()` `[pure virtual]`

Check whether the automaton contain an accepting run.

Return 0 if the automaton accept no run. Return an instance of [emptiness\\_check\\_result](#) otherwise. This instance might allow to obtain one sample acceptance run. The result has to be destroyed before the [emptiness\\_check](#) instance that generated it.

Some [emptiness\\_check](#) algorithms may allow [check\(\)](#) to be called several time, but generally you should not assume that.

Implemented in [spot::couvreur99\\_check](#), and [spot::couvreur99\\_check\\_shy](#).

**11.30.3.3** `virtual std::ostream& spot::emptiness_check::print_stats (std::ostream & os) const` `[virtual]`

Print statistics, if any.

Reimplemented in [spot::couvreur99\\_check](#).

### 11.30.4 Member Data Documentation

**11.30.4.1** `const tgba* spot::emptiness_check::a_` `[protected]`

The automaton.

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

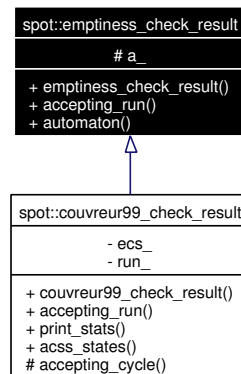
- [tgbaalgos/emptiness.hh](#)

## 11.31 `spot::emptiness_check_result` Class Reference

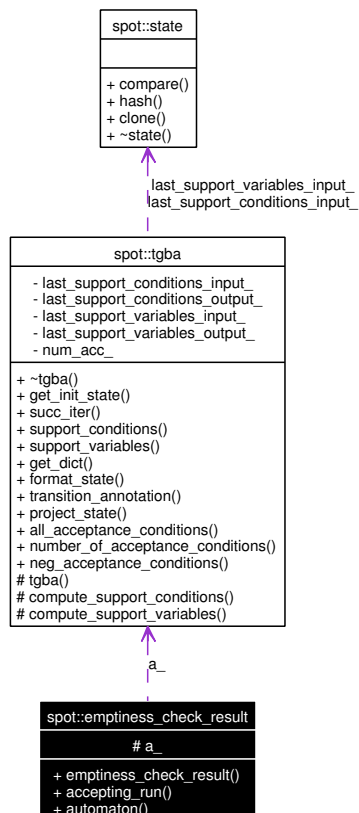
The result of an emptiness check.

```
#include <tgbaalgos/emptiness.hh>
```

Inheritance diagram for `spot::emptiness_check_result`:



Collaboration diagram for `spot::emptiness_check_result`:



## Public Member Functions

- `emptiness_check_result` (const `tgba` \*a)
- virtual `tgba_run` \* `accepting_run` ()  
*Return a run accepted by the automata passed to the emptiness check.*
- const `tgba` \* `automaton` () const

*The automaton on which an [accepting\\_run\(\)](#) was found.*

### Protected Attributes

- const [tgba](#) \* [a\\_](#)

*The automaton.*

### 11.31.1 Detailed Description

The result of an emptiness check.

Instances of these class should not last longer than the instances of [emptiness\\_check](#) that produced them as they may reference data internal to the check.

### 11.31.2 Constructor & Destructor Documentation

**11.31.2.1** [spot::emptiness\\_check\\_result::emptiness\\_check\\_result](#) (const [tgba](#) \* *a*) [inline]

### 11.31.3 Member Function Documentation

**11.31.3.1** virtual [tgba\\_run](#)\* [spot::emptiness\\_check\\_result::accepting\\_run](#) () [virtual]

Return a run accepted by the automata passed to the emptiness check.

This method might actually compute the acceptance run. (Not all emptiness check algorithms actually produce a counter-example as a side-effect of checking emptiness, some need some post-processing.)

This can also return 0 if the emptiness check algorithm cannot produce a counter example (that does not mean there is no counter-example; the mere existence of an instance of this class asserts the existence of a counter-example).

Reimplemented in [spot::couvreur99\\_check\\_result](#).

**11.31.3.2** const [tgba](#)\* [spot::emptiness\\_check\\_result::automaton](#) () const [inline]

The automaton on which an [accepting\\_run\(\)](#) was found.

### 11.31.4 Member Data Documentation

**11.31.4.1** const [tgba](#)\* [spot::emptiness\\_check\\_result::a\\_](#) [protected]

The automaton.

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

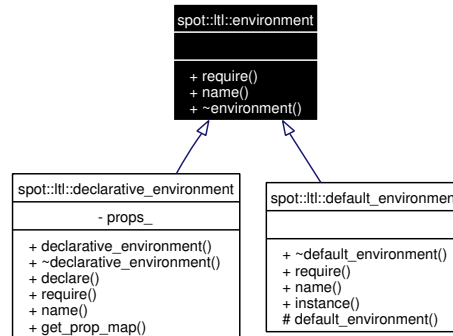
- [tgbaalgos/emptiness.hh](#)

## 11.32 spot::ltl::environment Class Reference

An environment that describes atomic propositions.

```
#include <ltlenv/environment.hh>
```

Inheritance diagram for spot::ltl::environment:



## Public Member Functions

- virtual [formula](#) \* [require](#) (const std::string &prop\_str)=0  
Obtain the formula associated to prop\_str.
- virtual const std::string & [name](#) ()=0  
Get the name of the environment.
- virtual [~environment](#) ()

### 11.32.1 Detailed Description

An environment that describes atomic propositions.

### 11.32.2 Constructor & Destructor Documentation

**11.32.2.1** virtual `spot::ltl::environment::~~environment` () [inline, virtual]

### 11.32.3 Member Function Documentation

**11.32.3.1** virtual const std::string& `spot::ltl::environment::name` () [pure virtual]

Get the name of the environment.

Implemented in [spot::ltl::declarative\\_environment](#), and [spot::ltl::default\\_environment](#).

**11.32.3.2** virtual [formula](#)\* `spot::ltl::environment::require` (const std::string & *prop\_str*) [pure virtual]

Obtain the formula associated to *prop\_str*.

Usually *prop\_str*, is the name of an atomic proposition, and `spot::ltl::require` simply returns the associated [spot::ltl::atomic\\_prop](#).

Note this is not a `const` method. Some environments will "create" the atomic proposition when requested.

We return a [spot::ltl::formula](#) instead of an [spot::ltl::atomic\\_prop](#), because this will allow nifty tricks (e.g., we could name formulae in an environment, and let the parser build a larger tree from these).

#### Returns:

0 iff *prop\_str* is not part of the environment, or the associated [spot::ltl::formula](#) otherwise.

Implemented in [spot::ltl::declarative\\_environment](#), and [spot::ltl::default\\_environment](#).

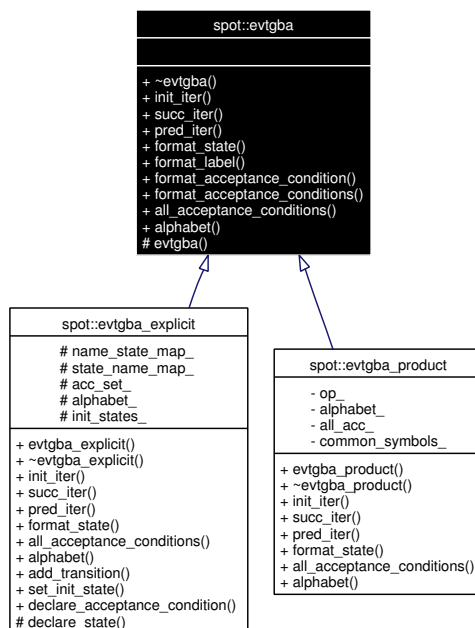
The documentation for this class was generated from the following file:

- [ltlenv/environment.hh](#)

## 11.33 spot::evtgba Class Reference

```
#include <evtgba/evtgba.hh>
```

Inheritance diagram for spot::evtgba:



#### Public Member Functions

- virtual [~evtgba](#) ()
- virtual [evtgba\\_iterator](#) \* [init\\_iter](#) () const =0
- virtual [evtgba\\_iterator](#) \* [succ\\_iter](#) (const [state](#) \*s) const =0
- virtual [evtgba\\_iterator](#) \* [pred\\_iter](#) (const [state](#) \*s) const =0
- virtual std::string [format\\_state](#) (const [state](#) \*state) const =0

*Format the state as a string for printing.*

- virtual std::string [format\\_label](#) (const [symbol](#) \*symbol) const
- virtual std::string [format\\_acceptance\\_condition](#) (const [symbol](#) \*symbol) const
- virtual std::string [format\\_acceptance\\_conditions](#) (const [symbol\\_set](#) &symset) const

- virtual const [symbol\\_set](#) & [all\\_acceptance\\_conditions](#) () const =0  
*Return the set of all acceptance conditions used by this automaton.*
- virtual const [symbol\\_set](#) & [alphabet](#) () const =0

### Protected Member Functions

- [evtgba](#) ()

### 11.33.1 Constructor & Destructor Documentation

**11.33.1.1** [spot::evtgba::evtgba](#) () [protected]

**11.33.1.2** virtual [spot::evtgba::~~evtgba](#) () [virtual]

### 11.33.2 Member Function Documentation

**11.33.2.1** virtual const [symbol\\_set](#)& [spot::evtgba::all\\_acceptance\\_conditions](#) () const [pure virtual]

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implemented in [spot::evtgba\\_explicit](#), and [spot::evtgba\\_product](#).

**11.33.2.2** virtual const [symbol\\_set](#)& [spot::evtgba::alphabet](#) () const [pure virtual]

Implemented in [spot::evtgba\\_explicit](#), and [spot::evtgba\\_product](#).

**11.33.2.3** virtual std::string [spot::evtgba::format\\_acceptance\\_condition](#) (const [symbol](#) \* *symbol*) const [virtual]

**11.33.2.4** virtual std::string [spot::evtgba::format\\_acceptance\\_conditions](#) (const [symbol\\_set](#) & *sym-set*) const [virtual]

**11.33.2.5** virtual std::string [spot::evtgba::format\\_label](#) (const [symbol](#) \* *symbol*) const [virtual]

**11.33.2.6** virtual std::string [spot::evtgba::format\\_state](#) (const [state](#) \* *state*) const [pure virtual]

Format the state as a string for printing.

This formatting is the responsibility of the automata who owns the state.

Implemented in [spot::evtgba\\_product](#).



**11.33.2.7** virtual [evtgba\\_iterator](#)\* [spot::evtgba::init\\_iter](#)() const [pure virtual]

Implemented in [spot::evtgba\\_explicit](#), and [spot::evtgba\\_product](#).

**11.33.2.8** virtual [evtgba\\_iterator](#)\* [spot::evtgba::pred\\_iter](#)(const [state](#) \* s) const [pure virtual]

Implemented in [spot::evtgba\\_product](#).

**11.33.2.9** virtual [evtgba\\_iterator](#)\* [spot::evtgba::succ\\_iter](#)(const [state](#) \* s) const [pure virtual]

Implemented in [spot::evtgba\\_product](#).

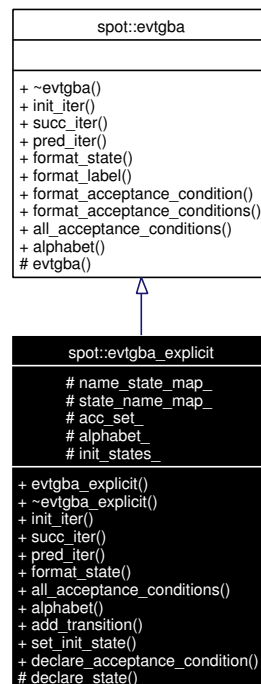
The documentation for this class was generated from the following file:

- [evtgba/evtgba.hh](#)

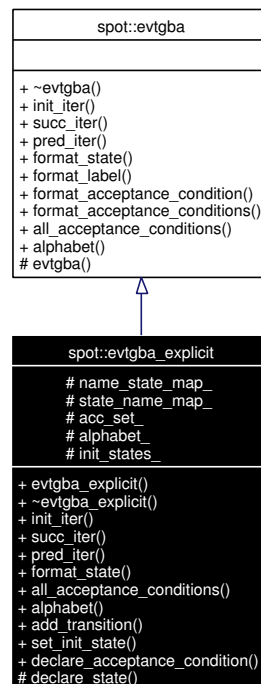
## 11.34 spot::evtgba\_explicit Class Reference

```
#include <evtgba/explicit.hh>
```

Inheritance diagram for [spot::evtgba\\_explicit](#):



Collaboration diagram for [spot::evtgba\\_explicit](#):



## Public Types

- typedef std::list< [transition](#) \* > [transition\\_list](#)

## Public Member Functions

- [evtgba\\_explicit](#) ()
- virtual [~evtgba\\_explicit](#) ()
- virtual [evtgba\\_iterator](#) \* [init\\_iter](#) () const
- virtual [evtgba\\_iterator](#) \* [succ\\_iter](#) (const [spot::state](#) \*s) const
- virtual [evtgba\\_iterator](#) \* [pred\\_iter](#) (const [spot::state](#) \*s) const
- virtual std::string [format\\_state](#) (const [spot::state](#) \*state) const
- virtual const [symbol\\_set](#) & [all\\_acceptance\\_conditions](#) () const  
*Return the set of all acceptance conditions used by this automaton.*
- virtual const [symbol\\_set](#) & [alphabet](#) () const
- [transition](#) \* [add\\_transition](#) (const std::string &source, const [rsymbol](#) &label, [rsymbol\\_set](#) acc, const std::string &dest)
- void [set\\_init\\_state](#) (const std::string &name)  
*Designate name as initial state.*
- void [declare\\_acceptance\\_condition](#) (const [rsymbol](#) &acc)
- virtual [evtgba\\_iterator](#) \* [succ\\_iter](#) (const [state](#) \*s) const =0
- virtual [evtgba\\_iterator](#) \* [pred\\_iter](#) (const [state](#) \*s) const =0
- virtual std::string [format\\_state](#) (const [state](#) \*state) const =0  
*Format the state as a string for printing.*

- virtual std::string [format\\_label](#) (const [symbol](#) \*symbol) const
- virtual std::string [format\\_acceptance\\_condition](#) (const [symbol](#) \*symbol) const
- virtual std::string [format\\_acceptance\\_conditions](#) (const [symbol\\_set](#) &symset) const

### Protected Types

- typedef Sgi::hash\_map< const std::string, [evtgba\\_explicit::state](#) \*, [string\\_hash](#) > [ns\\_map](#)
- typedef Sgi::hash\_map< const [evtgba\\_explicit::state](#) \*, std::string, [ptr\\_hash](#)< [evtgba\\_explicit::state](#) > > [sn\\_map](#)

### Protected Member Functions

- [state](#) \* [declare\\_state](#) (const std::string &name)

### Protected Attributes

- [ns\\_map](#) [name\\_state\\_map\\_](#)
- [sn\\_map](#) [state\\_name\\_map\\_](#)
- [symbol\\_set](#) [acc\\_set\\_](#)
- [symbol\\_set](#) [alphabet\\_](#)
- [transition\\_list](#) [init\\_states\\_](#)

### Classes

- struct [state](#)
- struct [transition](#)

*Explicit transitions (used by [spot::evtgba\\_explicit](#)).*

#### 11.34.1 Member Typedef Documentation

**11.34.1.1** typedef Sgi::hash\_map<const std::string, [evtgba\\_explicit::state](#)\*, [string\\_hash](#)> [spot::evtgba\\_explicit::ns\\_map](#) [protected]

**11.34.1.2** typedef Sgi::hash\_map<const [evtgba\\_explicit::state](#)\*, std::string, [ptr\\_hash](#)<[evtgba\\_explicit::state](#)> > [spot::evtgba\\_explicit::sn\\_map](#) [protected]

**11.34.1.3** typedef std::list<[transition](#)\*> [spot::evtgba\\_explicit::transition\\_list](#)

#### 11.34.2 Constructor & Destructor Documentation

**11.34.2.1** [spot::evtgba\\_explicit::evtgba\\_explicit](#) ()

**11.34.2.2** virtual [spot::evtgba\\_explicit::~~evtgba\\_explicit](#) () [virtual]

### 11.34.3 Member Function Documentation

**11.34.3.1** [transition\\*](#) `spot::evtgba_explicit::add_transition (const std::string & source, const rsymbol & label, rsymbol\_set acc, const std::string & dest)`

**11.34.3.2** `virtual const symbol\_set& spot::evtgba_explicit::all_acceptance_conditions () const` `[virtual]`

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these accepting conditions. I.e., the union of the accepting conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::evtgba](#).

**11.34.3.3** `virtual const symbol\_set& spot::evtgba_explicit::alphabet () const` `[virtual]`

Implements [spot::evtgba](#).

**11.34.3.4** `void spot::evtgba_explicit::declare_acceptance_condition (const rsymbol & acc)`

**11.34.3.5** `state* spot::evtgba_explicit::declare_state (const std::string & name)` `[protected]`

**11.34.3.6** `virtual std::string spot::evtgba::format_acceptance_condition (const symbol * symbol) const` `[virtual, inherited]`

**11.34.3.7** `virtual std::string spot::evtgba::format_acceptance_conditions (const symbol\_set & sym-set) const` `[virtual, inherited]`

**11.34.3.8** `virtual std::string spot::evtgba::format_label (const symbol * symbol) const` `[virtual, inherited]`

**11.34.3.9** `virtual std::string spot::evtgba::format_state (const state * state) const` `[pure virtual, inherited]`

Format the state as a string for printing.

This formatting is the responsibility of the automata who owns the state.

Implemented in [spot::evtgba\\_product](#).

**11.34.3.10** `virtual std::string spot::evtgba_explicit::format_state (const spot::state * state) const` `[virtual]`

**11.34.3.11** `virtual evtgba\_iterator* spot::evtgba_explicit::init_iter () const` `[virtual]`

Implements [spot::evtgba](#).

**11.34.3.12** virtual [evtgba\\_iterator](#)\* spot::evtgba::pred\_iter (const [state](#) \* s) const [pure virtual, inherited]

Implemented in [spot::evtgba\\_product](#).

**11.34.3.13** virtual [evtgba\\_iterator](#)\* spot::evtgba\_explicit::pred\_iter (const [spot::state](#) \* s) const [virtual]

**11.34.3.14** void spot::evtgba\_explicit::set\_init\_state (const std::string & name)

Designate *name* as initial state.

Can be called multiple times in case there is several initial states.

**11.34.3.15** virtual [evtgba\\_iterator](#)\* spot::evtgba::succ\_iter (const [state](#) \* s) const [pure virtual, inherited]

Implemented in [spot::evtgba\\_product](#).

**11.34.3.16** virtual [evtgba\\_iterator](#)\* spot::evtgba\_explicit::succ\_iter (const [spot::state](#) \* s) const [virtual]

#### 11.34.4 Member Data Documentation

**11.34.4.1** [symbol\\_set](#) spot::evtgba\_explicit::acc\_set\_ [protected]

**11.34.4.2** [symbol\\_set](#) spot::evtgba\_explicit::alphabet\_ [protected]

**11.34.4.3** [transition\\_list](#) spot::evtgba\_explicit::init\_states\_ [protected]

**11.34.4.4** [ns\\_map](#) spot::evtgba\_explicit::name\_state\_map\_ [protected]

**11.34.4.5** [sn\\_map](#) spot::evtgba\_explicit::state\_name\_map\_ [protected]

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

- [evtgba/explicit.hh](#)

#### 11.35 spot::evtgba\_explicit::state Struct Reference

```
#include <evtgba/explicit.hh>
```

##### Public Attributes

- [transition\\_list](#) in
- [transition\\_list](#) out

### 11.35.1 Member Data Documentation

#### 11.35.1.1 [transition\\_list](#) spot::evtgba\_explicit::state::in

#### 11.35.1.2 [transition\\_list](#) spot::evtgba\_explicit::state::out

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

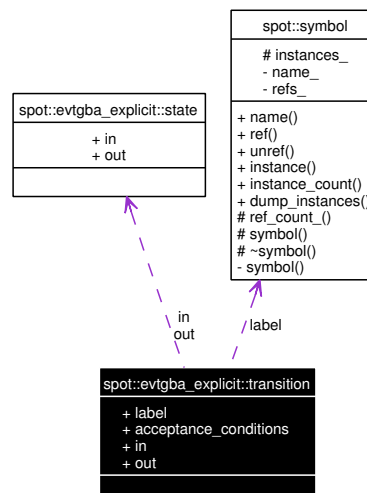
- [evtgba/explicit.hh](#)

## 11.36 spot::evtgba\_explicit::transition Struct Reference

Explicit transitions (used by [spot::evtgba\\_explicit](#)).

```
#include <evtgba/explicit.hh>
```

Collaboration diagram for spot::evtgba\_explicit::transition:



### Public Attributes

- const [symbol](#) \* [label](#)
- [symbol\\_set](#) [acceptance\\_conditions](#)
- [state](#) \* [in](#)
- [state](#) \* [out](#)

### 11.36.1 Detailed Description

Explicit transitions (used by [spot::evtgba\\_explicit](#)).

### 11.36.2 Member Data Documentation

#### 11.36.2.1 [symbol\\_set](#) spot::evtgba\_explicit::transition::acceptance\_conditions

11.36.2.2 [state\\*](#) [spot::evtgba\\_explicit::transition::in](#)

11.36.2.3 [const symbol\\*](#) [spot::evtgba\\_explicit::transition::label](#)

11.36.2.4 [state\\*](#) [spot::evtgba\\_explicit::transition::out](#)

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

- [evtgba/explicit.hh](#)

## 11.37 spot::evtgba\_iterator Class Reference

```
#include <evtgba/evtgbaiter.hh>
```

### Public Member Functions

- virtual [~evtgba\\_iterator](#) ()
- virtual void [first](#) ()=0
- virtual void [next](#) ()=0
- virtual bool [done](#) () const =0
- virtual const [state](#) \* [current\\_state](#) () const =0
- virtual const [symbol](#) \* [current\\_label](#) () const =0
- virtual [symbol\\_set](#) [current\\_acceptance\\_conditions](#) () const =0

### 11.37.1 Constructor & Destructor Documentation

11.37.1.1 virtual [spot::evtgba\\_iterator::~~evtgba\\_iterator](#) () [inline, virtual]

### 11.37.2 Member Function Documentation

11.37.2.1 virtual [symbol\\_set](#) [spot::evtgba\\_iterator::current\\_acceptance\\_conditions](#) () const [pure virtual]

11.37.2.2 virtual const [symbol\\*](#) [spot::evtgba\\_iterator::current\\_label](#) () const [pure virtual]

11.37.2.3 virtual const [state\\*](#) [spot::evtgba\\_iterator::current\\_state](#) () const [pure virtual]

11.37.2.4 virtual bool [spot::evtgba\\_iterator::done](#) () const [pure virtual]

11.37.2.5 virtual void [spot::evtgba\\_iterator::first](#) () [pure virtual]

11.37.2.6 virtual void [spot::evtgba\\_iterator::next](#) () [pure virtual]

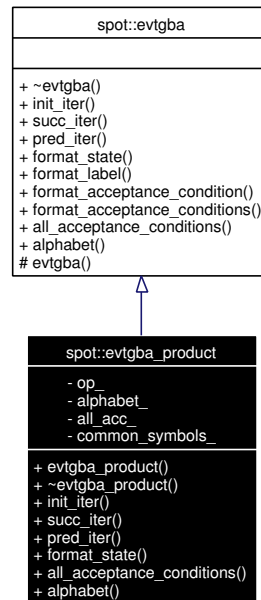
The documentation for this class was generated from the following file:

- [evtgba/evtgbaiter.hh](#)

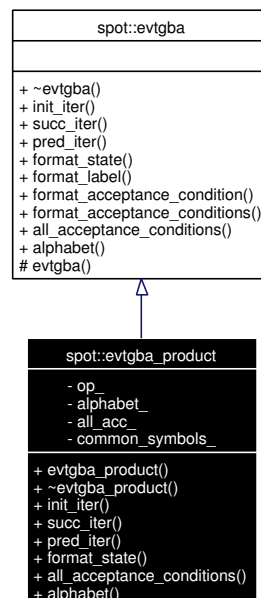
## 11.38 spot::evtgba\_product Class Reference

```
#include <evtgba/product.hh>
```

Inheritance diagram for spot::evtgba\_product:



Collaboration diagram for spot::evtgba\_product:





## Public Types

- typedef std::vector< const [evtgba \\*](#) > [evtgba\\_product\\_operands](#)
- typedef std::map< const [symbol \\*](#), std::set< int > > [common\\_symbol\\_table](#)

## Public Member Functions

- [evtgba\\_product](#) (const [evtgba\\_product\\_operands](#) &op)
- virtual [~evtgba\\_product](#) ()
- virtual [evtgba\\_iterator](#) \* [init\\_iter](#) () const
- virtual [evtgba\\_iterator](#) \* [succ\\_iter](#) (const [state](#) \*s) const
- virtual [evtgba\\_iterator](#) \* [pred\\_iter](#) (const [state](#) \*s) const
- virtual std::string [format\\_state](#) (const [state](#) \*state) const  
*Format the state as a string for printing.*
- virtual const [symbol\\_set](#) & [all\\_acceptance\\_conditions](#) () const  
*Return the set of all acceptance conditions used by this automaton.*
- virtual const [symbol\\_set](#) & [alphabet](#) () const
- virtual std::string [format\\_label](#) (const [symbol](#) \*symbol) const
- virtual std::string [format\\_acceptance\\_condition](#) (const [symbol](#) \*symbol) const
- virtual std::string [format\\_acceptance\\_conditions](#) (const [symbol\\_set](#) &symset) const

## Private Attributes

- const [evtgba\\_product\\_operands](#) op\_
- [symbol\\_set](#) alphabet\_
- [symbol\\_set](#) all\_acc\_
- [common\\_symbol\\_table](#) common\_symbols\_

### 11.38.1 Member Typedef Documentation

**11.38.1.1** typedef std::map<const [symbol](#)\*, std::set<int> > [spot::evtgba\\_product::common\\_symbol\\_table](#)

**11.38.1.2** typedef std::vector<const [evtgba](#)\*> [spot::evtgba\\_product::evtgba\\_product\\_operands](#)

### 11.38.2 Constructor & Destructor Documentation

**11.38.2.1** [spot::evtgba\\_product::evtgba\\_product](#) (const [evtgba\\_product\\_operands](#) & op)

**11.38.2.2** virtual [spot::evtgba\\_product::~~evtgba\\_product](#) () [virtual]

### 11.38.3 Member Function Documentation

**11.38.3.1** virtual const [symbol\\_set](#)& spot::evtgba\_product::all\_acceptance\_conditions () const [virtual]

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::evtgba](#).

**11.38.3.2** virtual const [symbol\\_set](#)& spot::evtgba\_product::alphabet () const [virtual]

Implements [spot::evtgba](#).

**11.38.3.3** virtual std::string spot::evtgba::format\_acceptance\_condition (const [symbol](#) \* *symbol*) const [virtual, inherited]

**11.38.3.4** virtual std::string spot::evtgba::format\_acceptance\_conditions (const [symbol\\_set](#) & *sym-set*) const [virtual, inherited]

**11.38.3.5** virtual std::string spot::evtgba::format\_label (const [symbol](#) \* *symbol*) const [virtual, inherited]

**11.38.3.6** virtual std::string spot::evtgba\_product::format\_state (const [state](#) \* *state*) const [virtual]

Format the state as a string for printing.

This formatting is the responsibility of the automata who owns the state.

Implements [spot::evtgba](#).

**11.38.3.7** virtual [evtgba\\_iterator](#)\* spot::evtgba\_product::init\_iter () const [virtual]

Implements [spot::evtgba](#).

**11.38.3.8** virtual [evtgba\\_iterator](#)\* spot::evtgba\_product::pred\_iter (const [state](#) \* *s*) const [virtual]

Implements [spot::evtgba](#).

**11.38.3.9** virtual [evtgba\\_iterator](#)\* spot::evtgba\_product::succ\_iter (const [state](#) \* *s*) const [virtual]

Implements [spot::evtgba](#).

### 11.38.4 Member Data Documentation

**11.38.4.1** [symbol\\_set](#) spot::evtgba\_product::all\_acc\_ [private]

11.38.4.2 [symbol\\_set spot::evtgba\\_product::alphabet\\_](#) [private]

11.38.4.3 [common\\_symbol\\_table spot::evtgba\\_product::common\\_symbols\\_](#) [private]

11.38.4.4 `const` [evtgba\\_product\\_operands spot::evtgba\\_product::op\\_](#) [private]

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

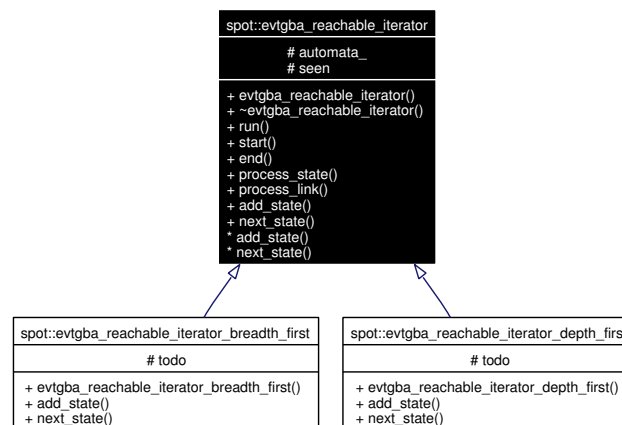
- [evtgba/product.hh](#)

## 11.39 spot::evtgba\_reachable\_iterator Class Reference

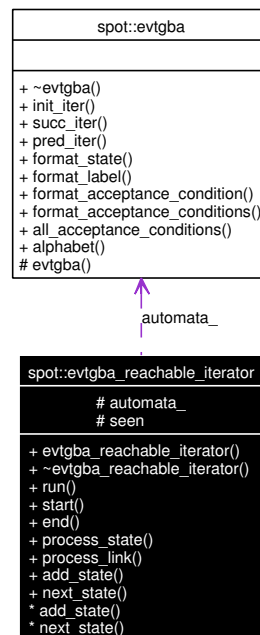
Iterate over all reachable states of a [spot::evtgba](#).

```
#include <evtgbaalgos/reachiter.hh>
```

Inheritance diagram for `spot::evtgba_reachable_iterator`:



Collaboration diagram for `spot::evtgba_reachable_iterator`:



## Public Member Functions

- [evtgba\\_reachable\\_iterator](#) (const [evtgba](#) \*a)
- virtual [~evtgba\\_reachable\\_iterator](#) ()
- void [run](#) ()  
*Iterate over all reachable states of a [spot::evtgba](#).*
- virtual void [start](#) (int n)  
*Called by [run\(\)](#) before starting its iteration.*
- virtual void [end](#) ()  
*Called by [run\(\)](#) once all states have been explored.*
- virtual void [process\\_state](#) (const [state](#) \*s, int n, [evtgba\\_iterator](#) \*si)
- virtual void [process\\_link](#) (int in, int out, const [evtgba\\_iterator](#) \*si)

## Todo list management.

Called by [run\(\)](#) to register newly discovered states.

[spot::evtgba\\_reachable\\_iterator\\_depth\\_first](#) and [spot::evtgba\\_reachable\\_iterator\\_breadth\\_first](#) offer two precanned implementations for these functions.

- virtual void [add\\_state](#) (const [state](#) \*s)=0
- virtual const [state](#) \* [next\\_state](#) ()=0  
*Called by [run\(\)](#) to obtain the.*

## Protected Types

- typedef Sgi::hash\_map< const [state](#) \*, int, [state\\_ptr\\_hash](#), [state\\_ptr\\_equal](#) > [seen\\_map](#)

### Protected Attributes

- const [evtgba](#) \* [automata\\_](#)  
*The [spot::evtgba](#) to explore.*
- [seen\\_map](#) seen  
*States already seen.*

### 11.39.1 Detailed Description

Iterate over all reachable states of a [spot::evtgba](#).

### 11.39.2 Member Typedef Documentation

**11.39.2.1** `typedef Sgi::hash_map<const state*, int, state\_ptr\_hash, state\_ptr\_equal> spot::evtgba\_reachable\_iterator::seen\_map [protected]`

### 11.39.3 Constructor & Destructor Documentation

**11.39.3.1** `spot::evtgba_reachable_iterator::evtgba_reachable_iterator (const evtgba * a)`

**11.39.3.2** `virtual spot::evtgba_reachable_iterator::~~evtgba\_reachable\_iterator () [virtual]`

### 11.39.4 Member Function Documentation

**11.39.4.1** `virtual void spot::evtgba_reachable_iterator::add_state (const state * s) [pure virtual]`

Implemented in [spot::evtgba\\_reachable\\_iterator\\_depth\\_first](#), and [spot::evtgba\\_reachable\\_iterator\\_breadth\\_first](#).

**11.39.4.2** `virtual void spot::evtgba_reachable_iterator::end () [virtual]`

Called by [run\(\)](#) once all states have been explored.

**11.39.4.3** `virtual const state* spot::evtgba_reachable_iterator::next_state () [pure virtual]`

Called by [run\(\)](#) to obtain the.

Implemented in [spot::evtgba\\_reachable\\_iterator\\_depth\\_first](#), and [spot::evtgba\\_reachable\\_iterator\\_breadth\\_first](#).

**11.39.4.4** `virtual void spot::evtgba_reachable_iterator::process_link (int in, int out, const evtgba\_iterator * si) [virtual]`

Called by [run\(\)](#) to process a transition.

**Parameters:**

- in* The source state number.
- out* The destination state number.
- si* The `spot::evtgba_iterator` positionned on the current transition.

**11.39.4.5** `virtual void spot::evtgba_reachable_iterator::process_state (const state * s, int n, evtgba\_iterator * si)` [virtual]

Called by `run()` to process a state.

**Parameters:**

- s* The current state.
- n* An unique number assigned to *s*.
- si* The `spot::evtgba_iterator` for *s*.

**11.39.4.6** `void spot::evtgba_reachable_iterator::run ()`

Iterate over all reachable states of a `spot::evtgba`.

This is a template method that will call `add_state()`, `next_state()`, `start()`, `end()`, `process_state()`, and `process_link()`, while it iterate over state.

**11.39.4.7** `virtual void spot::evtgba_reachable_iterator::start (int n)` [virtual]

Called by `run()` before starting its iteration.

**Parameters:**

- n* The number of initial states.

**11.39.5 Member Data Documentation**

**11.39.5.1** `const evtgba* spot::evtgba_reachable_iterator::automata_` [protected]

The `spot::evtgba` to explore.

**11.39.5.2** `seen\_map spot::evtgba_reachable_iterator::seen` [protected]

States already seen.

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

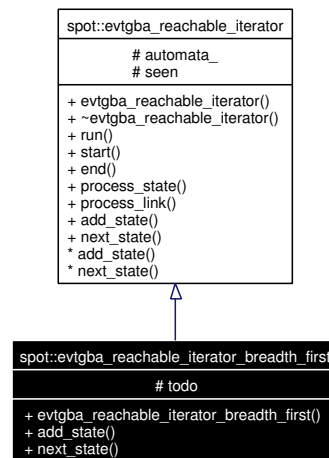
- `evtgbaaalgos/reachiter.hh`

**11.40 `spot::evtgba_reachable_iterator_breadth_first` Class Reference**

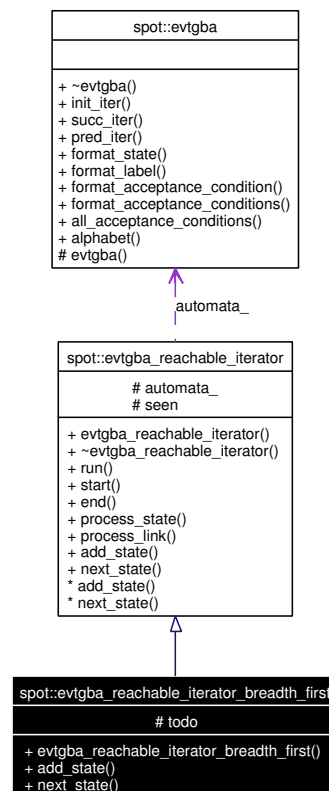
An implementation of `spot::evtgba_reachable_iterator` that browses states breadth first.

```
#include <evtgbaaalgos/reachiter.hh>
```

Inheritance diagram for `spot::evtgba_reachable_iterator_breadth_first`:



Collaboration diagram for `spot::evtgba_reachable_iterator_breadth_first`:



## Public Member Functions

- `evtgba_reachable_iterator_breadth_first` (const `evtgba` \*a)
- virtual void `add_state` (const `state` \*s)
- virtual const `state` \* `next_state` ()

*Called by `run()` to obtain the.*

- void [run](#) ()  
*Iterate over all reachable states of a [spot::evtgba](#).*
- virtual void [start](#) (int n)  
*Called by [run\(\)](#) before starting its iteration.*
- virtual void [end](#) ()  
*Called by [run\(\)](#) once all states have been explored.*
- virtual void [process\\_state](#) (const [state](#) \*s, int n, [evtgba\\_iterator](#) \*si)
- virtual void [process\\_link](#) (int in, int out, const [evtgba\\_iterator](#) \*si)

### Protected Types

- typedef Sgi::hash\_map< const [state](#) \*, int, [state\\_ptr\\_hash](#), [state\\_ptr\\_equal](#) > [seen\\_map](#)

### Protected Attributes

- std::deque< const [state](#) \* > [todo](#)  
*A queue of states yet to explore.*
- const [evtgba](#) \* [automata\\_](#)  
*The [spot::evtgba](#) to explore.*
- [seen\\_map](#) [seen](#)  
*States already seen.*

## 11.40.1 Detailed Description

An implementation of [spot::evtgba\\_reachable\\_iterator](#) that browses states breadth first.

## 11.40.2 Member Typedef Documentation

**11.40.2.1** typedef Sgi::hash\_map<const [state](#)\*, int, [state\\_ptr\\_hash](#), [state\\_ptr\\_equal](#)> [spot::evtgba\\_reachable\\_iterator::seen\\_map](#) [protected, inherited]

## 11.40.3 Constructor & Destructor Documentation

**11.40.3.1** [spot::evtgba\\_reachable\\_iterator\\_breadth\\_first::evtgba\\_reachable\\_iterator\\_breadth\\_first](#) (const [evtgba](#) \* a)

## 11.40.4 Member Function Documentation

**11.40.4.1** virtual void [spot::evtgba\\_reachable\\_iterator\\_breadth\\_first::add\\_state](#) (const [state](#) \* s) [virtual]

Implements [spot::evtgba\\_reachable\\_iterator](#).



**11.40.4.2 virtual void spot::evtgba\_reachable\_iterator::end ()** [virtual, inherited]

Called by [run\(\)](#) once all states have been explored.

**11.40.4.3 virtual const state\* spot::evtgba\_reachable\_iterator\_breadth\_first::next\_state ()** [virtual]

Called by [run\(\)](#) to obtain the.

Implements [spot::evtgba\\_reachable\\_iterator](#).

**11.40.4.4 virtual void spot::evtgba\_reachable\_iterator::process\_link (int *in*, int *out*, const evtgba\_iterator \* *si*)** [virtual, inherited]

Called by [run\(\)](#) to process a transition.

**Parameters:**

*in* The source state number.

*out* The destination state number.

*si* The [spot::evtgba\\_iterator](#) positionned on the current transition.

**11.40.4.5 virtual void spot::evtgba\_reachable\_iterator::process\_state (const state \* *s*, int *n*, evtgba\_iterator \* *si*)** [virtual, inherited]

Called by [run\(\)](#) to process a state.

**Parameters:**

*s* The current state.

*n* An unique number assigned to *s*.

*si* The [spot::evtgba\\_iterator](#) for *s*.

**11.40.4.6 void spot::evtgba\_reachable\_iterator::run ()** [inherited]

Iterate over all reachable states of a [spot::evtgba](#).

This is a template method that will call [add\\_state\(\)](#), [next\\_state\(\)](#), [start\(\)](#), [end\(\)](#), [process\\_state\(\)](#), and [process\\_link\(\)](#), while it iterate over state.

**11.40.4.7 virtual void spot::evtgba\_reachable\_iterator::start (int *n*)** [virtual, inherited]

Called by [run\(\)](#) before starting its iteration.

**Parameters:**

*n* The number of initial states.

**11.40.5 Member Data Documentation****11.40.5.1 const evtgba\* spot::evtgba\_reachable\_iterator::automata\_** [protected, inherited]

The [spot::evtgba](#) to explore.

**11.40.5.2** `seen_map` `spot::evtgba_reachable_iterator::seen` [protected, inherited]

States already seen.

**11.40.5.3** `std::deque<const state*>` `spot::evtgba_reachable_iterator_breadth_first::todo` [protected]

A queue of states yet to explore.

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

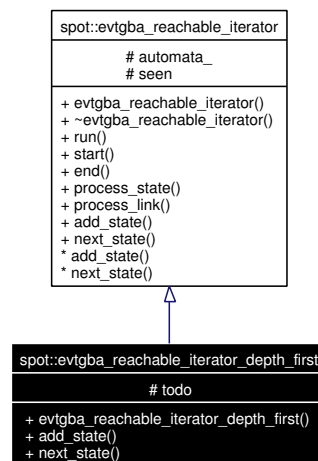
- `evtgbaalgos/reachiter.hh`

**11.41** `spot::evtgba_reachable_iterator_depth_first` Class Reference

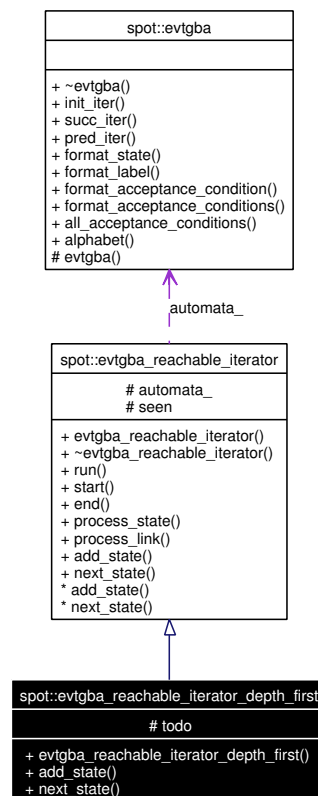
An implementation of `spot::evtgba_reachable_iterator` that browses states depth first.

```
#include <evtgbaalgos/reachiter.hh>
```

Inheritance diagram for `spot::evtgba_reachable_iterator_depth_first`:



Collaboration diagram for `spot::evtgba_reachable_iterator_depth_first`:



## Public Member Functions

- `evtgba_reachable_iterator_depth_first` (const `evtgba` \*a)
- virtual void `add_state` (const `state` \*s)
- virtual const `state` \* `next_state` ()  
*Called by `run()` to obtain the.*
- void `run` ()  
*Iterate over all reachable states of a `spot::evtgba`.*
- virtual void `start` (int n)  
*Called by `run()` before starting its iteration.*
- virtual void `end` ()  
*Called by `run()` once all states have been explored.*
- virtual void `process_state` (const `state` \*s, int n, `evtgba_iterator` \*si)
- virtual void `process_link` (int in, int out, const `evtgba_iterator` \*si)

## Protected Types

- typedef `Sgi::hash_map`< const `state` \*, int, `state_ptr_hash`, `state_ptr_equal` > `seen_map`

## Protected Attributes

- `std::stack< const state * > todo`  
*A stack of states yet to explore.*
- `const evtgba * automata_`  
*The spot::evtgba to explore.*
- `seen_map seen`  
*States already seen.*

### 11.41.1 Detailed Description

An implementation of `spot::evtgba_reachable_iterator` that browses states depth first.

### 11.41.2 Member Typedef Documentation

**11.41.2.1** `typedef Sgi::hash_map<const state*, int, state_ptr_hash, state_ptr_equal> spot::evtgba_reachable_iterator::seen_map` [protected, inherited]

### 11.41.3 Constructor & Destructor Documentation

**11.41.3.1** `spot::evtgba_reachable_iterator_depth_first::evtgba_reachable_iterator_depth_first (const evtgba * a)`

### 11.41.4 Member Function Documentation

**11.41.4.1** `virtual void spot::evtgba_reachable_iterator_depth_first::add_state (const state * s)` [virtual]

Implements `spot::evtgba_reachable_iterator`.

**11.41.4.2** `virtual void spot::evtgba_reachable_iterator::end ()` [virtual, inherited]

Called by `run()` once all states have been explored.

**11.41.4.3** `virtual const state* spot::evtgba_reachable_iterator_depth_first::next_state ()` [virtual]

Called by `run()` to obtain the.

Implements `spot::evtgba_reachable_iterator`.

**11.41.4.4** `virtual void spot::evtgba_reachable_iterator::process_link (int in, int out, const evtgba_iterator * si)` [virtual, inherited]

Called by `run()` to process a transition.

#### Parameters:

*in* The source state number.

*out* The destination state number.

*si* The `spot::evtgba_iterator` positionned on the current transition.

**11.41.4.5** `virtual void spot::evtgba_reachable_iterator::process_state (const state * s, int n, evtgba\_iterator * si)` [virtual, inherited]

Called by `run()` to process a state.

**Parameters:**

*s* The current state.

*n* An unique number assigned to *s*.

*si* The `spot::evtgba_iterator` for *s*.

**11.41.4.6** `void spot::evtgba_reachable_iterator::run ()` [inherited]

Iterate over all reachable states of a `spot::evtgba`.

This is a template method that will call `add_state()`, `next_state()`, `start()`, `end()`, `process_state()`, and `process_link()`, while it iterate over state.

**11.41.4.7** `virtual void spot::evtgba_reachable_iterator::start (int n)` [virtual, inherited]

Called by `run()` before starting its iteration.

**Parameters:**

*n* The number of initial states.

## 11.41.5 Member Data Documentation

**11.41.5.1** `const evtgba* spot::evtgba\_reachable\_iterator::automata\_` [protected, inherited]

The `spot::evtgba` to explore.

**11.41.5.2** `seen\_map spot::evtgba\_reachable\_iterator::seen` [protected, inherited]

States already seen.

**11.41.5.3** `std::stack<const state*> spot::evtgba\_reachable\_iterator\_depth\_first::todo` [protected]

A stack of states yet to explore.

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

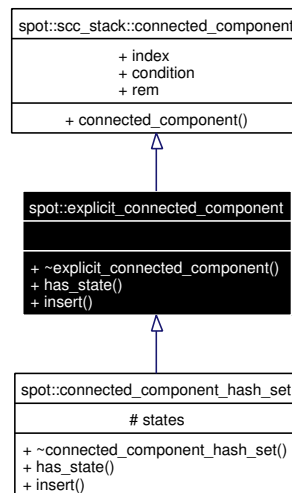
- `evtgbalgorithms/reachiter.hh`

## 11.42 spot::explicit\_connected\_component Class Reference

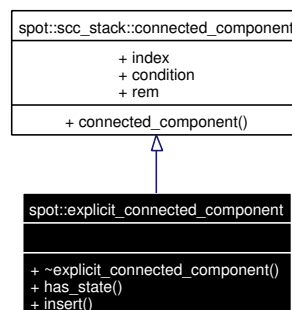
An SCC storing all its states explicitly.

```
#include <tgbaalgos/gtec/explscs.h>
```

Inheritance diagram for spot::explicit\_connected\_component:



Collaboration diagram for spot::explicit\_connected\_component:



### Public Member Functions

- virtual `~explicit_connected_component()`
- virtual const `state * has_state (const state *s) const =0`  
*Check if the SCC contains states s.*
- virtual void `insert (const state *s)=0`  
*Insert a new state in the SCC.*

### Public Attributes

- int `index`

*Index of the SCC.*

- `bdd` [condition](#)
- `std::list< const state * >` [rem](#)

### 11.42.1 Detailed Description

An SCC storing all its states explicitly.

### 11.42.2 Constructor & Destructor Documentation

**11.42.2.1** `virtual` `spot::explicit_connected_component::~explicit\_connected\_component` ()  
[`inline`, `virtual`]

### 11.42.3 Member Function Documentation

**11.42.3.1** `virtual const state* spot::explicit_connected_component::has_state (const state * s) const`  
[`pure virtual`]

Check if the SCC contains states *s*.

Return the representative of *s* in the SCC, and delete *s* if it is different (acting like `numbered_state_heap::filter`), or 0 otherwise.

Implemented in [spot::connected\\_component\\_hash\\_set](#).

**11.42.3.2** `virtual void spot::explicit_connected_component::insert (const state * s)` [`pure virtual`]

Insert a new state in the SCC.

Implemented in [spot::connected\\_component\\_hash\\_set](#).

### 11.42.4 Member Data Documentation

**11.42.4.1** `bdd spot::scc\_stack::connected\_component::condition` [`inherited`]

The bdd condition is the union of all acceptance conditions of transitions which connect the states of the connected component.

**11.42.4.2** `int spot::scc\_stack::connected\_component::index` [`inherited`]

Index of the SCC.

**11.42.4.3** `std::list<const state*> spot::scc\_stack::connected\_component::rem` [`inherited`]

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

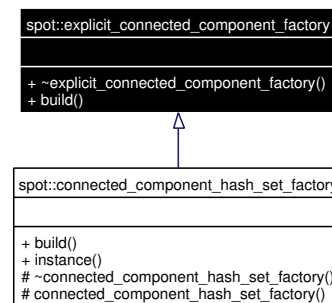
- `tgbaalgos/gtec/explscc.hh`

## 11.43 spot::explicit\_connected\_component\_factory Class Reference

Abstract factory for [explicit\\_connected\\_component](#).

```
#include <tgbalgorithms/gtec/explsc.h>
```

Inheritance diagram for spot::explicit\_connected\_component\_factory:



### Public Member Functions

- virtual [~explicit\\_connected\\_component\\_factory](#) ()
- virtual [explicit\\_connected\\_component](#) \* [build](#) () const =0  
Create an [explicit\\_connected\\_component](#).

### 11.43.1 Detailed Description

Abstract factory for [explicit\\_connected\\_component](#).

### 11.43.2 Constructor & Destructor Documentation

**11.43.2.1** virtual `spot::explicit_connected_component_factory::~explicit_connected_component_factory` () [inline, virtual]

### 11.43.3 Member Function Documentation

**11.43.3.1** virtual `explicit_connected_component*` `spot::explicit_connected_component_factory::build` () const [pure virtual]

Create an [explicit\\_connected\\_component](#).

Implemented in [spot::connected\\_component\\_hash\\_set\\_factory](#).

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

- `tgbalgorithms/gtec/explsc.h`

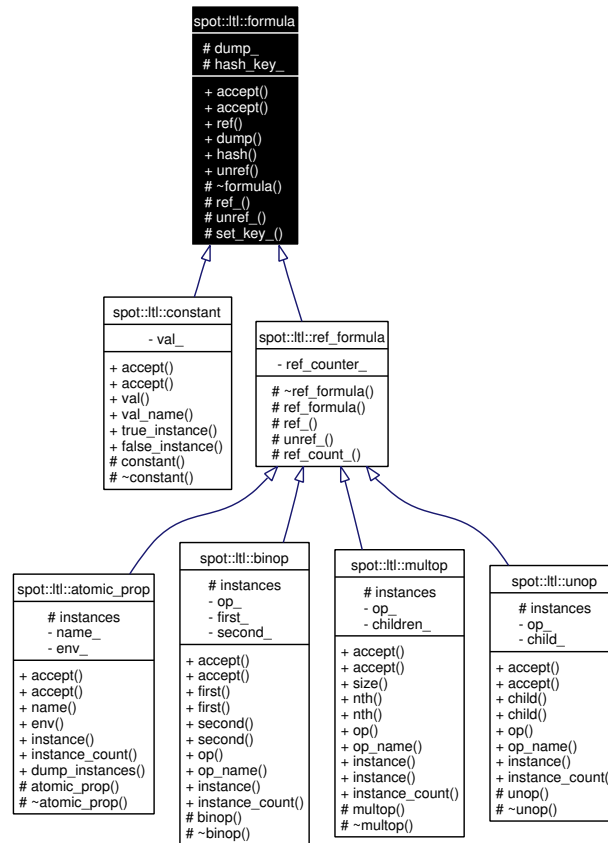
## 11.44 spot::ltl::formula Class Reference

An LTL formula.



```
#include <ltlast/formula.hh>
```

Inheritance diagram for spot::ltl::formula:



## Public Member Functions

- virtual void **accept** (visitor &v)=0  
*Entry point for vsport::ltl::visitor instances.*
- virtual void **accept** (const\_visitor &v) const =0  
*Entry point for vsport::ltl::const\_visitor instances.*
- formula \* ref** ()  
*clone this node*
- const std::string & **dump** () const  
*Return a canonic representation of the formula.*
- const size\_t **hash** () const  
*Return a hash\_key for the formula.*

### Static Public Member Functions

- static void `unref` (`formula *f`)  
*release this node*

### Protected Member Functions

- virtual `~formula` ()
- virtual void `ref_` ()  
*increment reference counter if any*
- virtual bool `unref_` ()  
*decrement reference counter if any, return true when the instance must be deleted (usually when the counter hits 0).*
- void `set_key_` ()  
*Compute key\_ from dump\_.*

### Protected Attributes

- std::string `dump_`  
*The canonic representation of the formula.*
- size\_t `hash_key_`  
*The hash key of this formula.*

#### 11.44.1 Detailed Description

An LTL formula.

The only way you can work with a formula is to build a `spot::ltl::visitor` or `spot::ltl::const_visitor`.

#### 11.44.2 Constructor & Destructor Documentation

**11.44.2.1** virtual `spot::ltl::formula::~~formula` () [protected, virtual]

#### 11.44.3 Member Function Documentation

**11.44.3.1** virtual void `spot::ltl::formula::accept` (`const_visitor &v`) const [pure virtual]

Entry point for `vspot::ltl::const_visitor` instances.

Implemented in `spot::ltl::atomic_prop`, `spot::ltl::binop`, `spot::ltl::constant`, `spot::ltl::multop`, and `spot::ltl::unop`.

**11.44.3.2** `virtual void spot::ltl::formula::accept (visitor & v) [pure virtual]`

Entry point for `vspot::ltl::visitor` instances.

Implemented in `spot::ltl::atomic_prop`, `spot::ltl::binop`, `spot::ltl::constant`, `spot::ltl::multop`, and `spot::ltl::unop`.

**11.44.3.3** `const std::string& spot::ltl::formula::dump () const`

Return a canonic representation of the formula.

**11.44.3.4** `const size_t spot::ltl::formula::hash () const [inline]`

Return a `hash_key` for the formula.

**11.44.3.5** `formula* spot::ltl::formula::ref ()`

clone this node

This increments the reference counter of this node (if one is used). You should almost never use this method directly as it doesn't touch the children. If you want to clone a whole formula, use `spot::ltl::clone()` instead.

**11.44.3.6** `virtual void spot::ltl::formula::ref_ () [protected, virtual]`

increment reference counter if any

Reimplemented in `spot::ltl::ref_formula`.

**11.44.3.7** `void spot::ltl::formula::set_key_ () [protected]`

Compute `key_` from `dump_`.

Should be called once in each object, after `dump_` has been set.

**11.44.3.8** `static void spot::ltl::formula::unref (formula * f) [static]`

release this node

This decrements the reference counter of this node (if one is used) and can free the object. You should almost never use this method directly as it doesn't touch the children. If you want to release a whole formula, use `spot::ltl::destroy()` instead.

**11.44.3.9** `virtual bool spot::ltl::formula::unref_ () [protected, virtual]`

decrement reference counter if any, return true when the instance must be deleted (usually when the counter hits 0).

Reimplemented in `spot::ltl::ref_formula`.

**11.44.4** Member Data Documentation**11.44.4.1** `std::string spot::ltl::formula::dump_ [protected]`

The canonic representation of the formula.

#### 11.44.4.2 `size_t spot::ltl::formula::hash_key_` [protected]

The hash key of this formula.

Initialized by `set_key_()`.

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

- `ltlast/formula.hh`

### 11.45 `spot::ltl::formula_ptr_hash` Struct Reference

Hash Function for `const formula*`.

```
#include <ltlast/formula.hh>
```

#### Public Member Functions

- `size_t operator()` (`const formula *that`) `const`

#### 11.45.1 Detailed Description

Hash Function for `const formula*`.

This is meant to be used as a hash functor for Sgi's `hash_map` whose key are of type `const formula*`.

For instance here is how one could declare a map of `const::formula*`.

```
// Remember how many times each formula has been seen.
Sgi::hash_map<const spot::ltl::formula*, int,
             const spot::ltl::formula_ptr_hash> seen;
```

#### 11.45.2 Member Function Documentation

##### 11.45.2.1 `size_t spot::ltl::formula_ptr_hash::operator()` (`const formula *that`) `const` [inline]

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

- `ltlast/formula.hh`

### 11.46 `spot::ltl::formula_ptr_less_than` Struct Reference

Strict Weak Ordering for `const formula*`.

```
#include <ltlast/formula.hh>
```

#### Public Member Functions

- `bool operator()` (`const formula *left`, `const formula *right`) `const`

### 11.46.1 Detailed Description

Strict Weak Ordering for `const formula*`.

This is meant to be used as a comparison functor for STL map whose key are of type `const formula*`.

For instance here is how one could declare a map of `const :: formula*`.

```
// Remember how many times each formula has been seen.
std::map<const spot::ltl::formula*, int,
        spot::formula_ptr_less_than> seen;
```

### 11.46.2 Member Function Documentation

**11.46.2.1** `bool spot::ltl::formula_ptr_less_than::operator() (const formula * left, const formula * right) const` [inline]

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

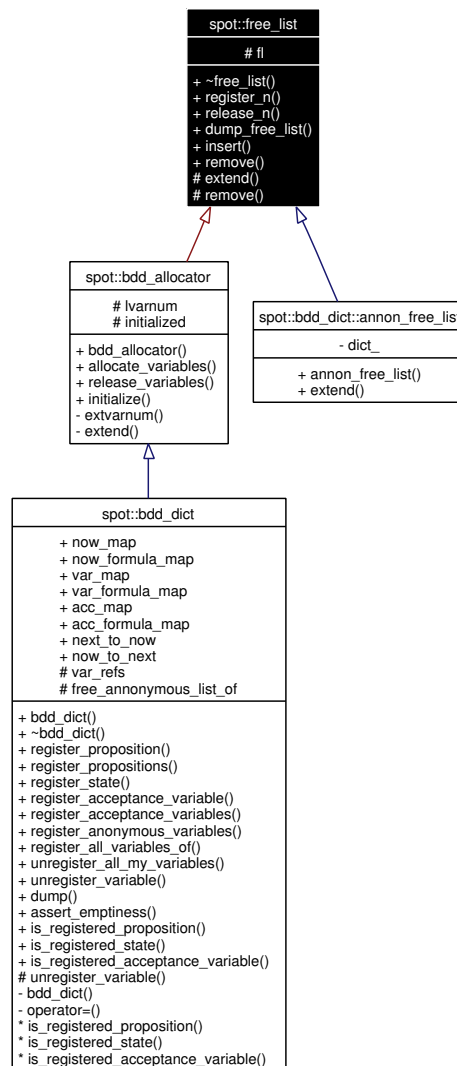
- `ltlast/formula.hh`

## 11.47 `spot::free_list` Class Reference

Manage list of free integers.

```
#include <misc/freelist.hh>
```

Inheritance diagram for `spot::free_list`:



## Public Member Functions

- virtual `~free_list()`
- int `register_n` (int n)  
*Find n consecutive integers.*
- void `release_n` (int base, int n)  
*Release n consecutive integers starting at base.*
- std::ostream & `dump_free_list` (std::ostream &os) const  
*Dump the list to os for debugging.*
- void `insert` (int base, int n)  
*Extend the list by inserting a new pos-length pair.*
- void `remove` (int base, int n=0)

*Remove n consecutive entries from the list, starting at base.*

### Protected Types

- typedef std::pair< int, int > `pos_lenght_pair`  
Such pairs describe second free integer starting at first.
- typedef std::list< `pos_lenght_pair` > `free_list_type`

### Protected Member Functions

- virtual int `extend` (int n)=0
- void `remove` (free\_list\_type::iterator i, int base, int n)  
*Remove n consecutive entries from the list, starting at base.*

### Protected Attributes

- `free_list_type` fl  
*Tracks unused BDD variables.*

#### 11.47.1 Detailed Description

Manage list of free integers.

#### 11.47.2 Member Typedef Documentation

**11.47.2.1** typedef std::list<`pos_lenght_pair`> `spot::free_list::free_list_type` [protected]

**11.47.2.2** typedef std::pair<int, int> `spot::free_list::pos_lenght_pair` [protected]

Such pairs describe second free integer starting at first.

#### 11.47.3 Constructor & Destructor Documentation

**11.47.3.1** virtual `spot::free_list::~free_list` () [virtual]

#### 11.47.4 Member Function Documentation

**11.47.4.1** std::ostream& `spot::free_list::dump_free_list` (std::ostream & os) const

Dump the list to os for debugging.

**11.47.4.2** `virtual int spot::free_list::extend (int n)` [protected, pure virtual]

Allocate *n* integer.

This function is called by `register_n()` when the free list is empty or if *n* consecutive integers could not be found. It should allocate more integers, possibly changing the list, and return the first integer on a range of *n* consecutive integer requested by the user.

Implemented in `spot::bdd_allocator`, and `spot::bdd_dict::annon_free_list`.

**11.47.4.3** `void spot::free_list::insert (int base, int n)`

Extend the list by inserting a new pos-length pair.

**11.47.4.4** `int spot::free_list::register_n (int n)`

Find *n* consecutive integers.

Browse the list of free integers until *n* consecutive integers are found. Extend the list (using `extend()`) otherwise.

**Returns:**

the first integer of the range

**11.47.4.5** `void spot::free_list::release_n (int base, int n)`

Release *n* consecutive integers starting at *base*.

**11.47.4.6** `void spot::free_list::remove (free_list_type::iterator i, int base, int n)` [protected]

Remove *n* consecutive entries from the list, starting at *base*.

**11.47.4.7** `void spot::free_list::remove (int base, int n = 0)`

Remove *n* consecutive entries from the list, starting at *base*.

**11.47.5** Member Data Documentation**11.47.5.1** `free_list_type spot::free_list::fl` [protected]

Tracks unused BDD variables.

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

- `misc/freelist.hh`

**11.48** `spot::gspn_exception` Class Reference

An exception used to forward GSPN errors.

```
#include <gspn/common.hh>
```



### Public Member Functions

- [gspn\\_exeption](#) (const std::string &where, int err)
- int [get\\_err](#) () const
- std::string [get\\_where](#) () const

### Private Attributes

- int [err\\_](#)
- std::string [where\\_](#)

#### 11.48.1 Detailed Description

An exeption used to forward GSPN errors.

#### 11.48.2 Constructor & Destructor Documentation

**11.48.2.1** `spot::gspn_exeption::gspn_exeption (const std::string & where, int err)` [inline]

#### 11.48.3 Member Function Documentation

**11.48.3.1** `int spot::gspn_exeption::get_err () const` [inline]

**11.48.3.2** `std::string spot::gspn_exeption::get_where () const` [inline]

#### 11.48.4 Member Data Documentation

**11.48.4.1** `int spot::gspn\_exeption::err\_` [private]

**11.48.4.2** `std::string spot::gspn\_exeption::where\_` [private]

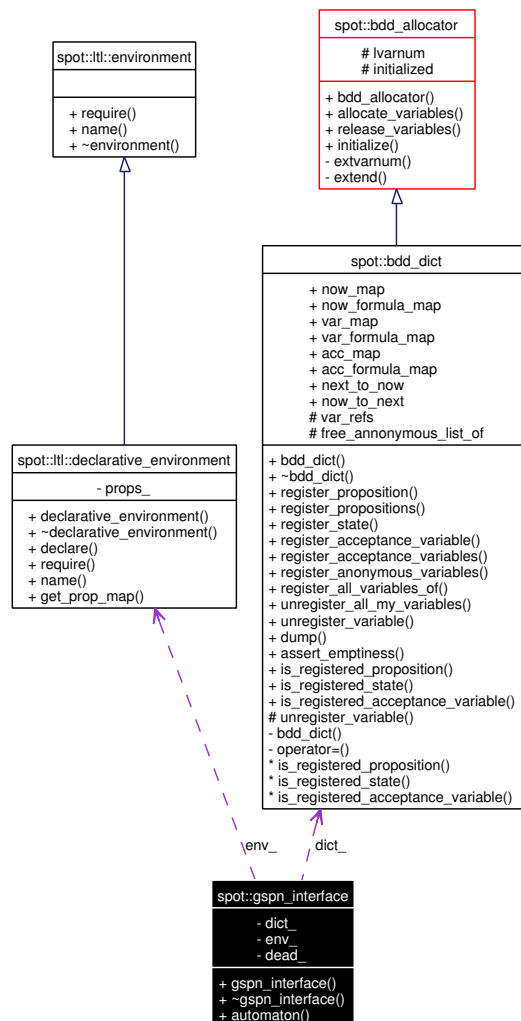
The documentation for this class was generated from the following file:

- [gspn/common.hh](#)

## 11.49 spot::gspn\_interface Class Reference

```
#include <gspn/gspn.hh>
```

Collaboration diagram for spot::gspn\_interface:



## Public Member Functions

- `gspn_interface` (int argc, char \*\*argv, `bdd_dict` \*dict, `ltl::declarative_environment` &env, const std::string &dead="true")
- `~gspn_interface` ()
- `tgba` \* `automaton` () const

## Private Attributes

- `bdd_dict` \* `dict_`
- `ltl::declarative_environment` & `env_`
- const std::string `dead_`

### 11.49.1 Constructor & Destructor Documentation

**11.49.1.1** `spot::gspn_interface::gspn_interface` (int *argc*, char \*\* *argv*, `bdd_dict` \* *dict*, `ltl::declarative_environment` & *env*, const std::string & *dead* = "true")

11.49.1.2 spot::gspn\_interface::~~gspn\_interface ()

### 11.49.2 Member Function Documentation

11.49.2.1 tgba\* spot::gspn\_interface::automaton () const

### 11.49.3 Member Data Documentation

11.49.3.1 const std::string spot::gspn\_interface::dead\_ [private]

11.49.3.2 bdd\_dict\* spot::gspn\_interface::dict\_ [private]

11.49.3.3 ltl::declarative\_environment& spot::gspn\_interface::env\_ [private]

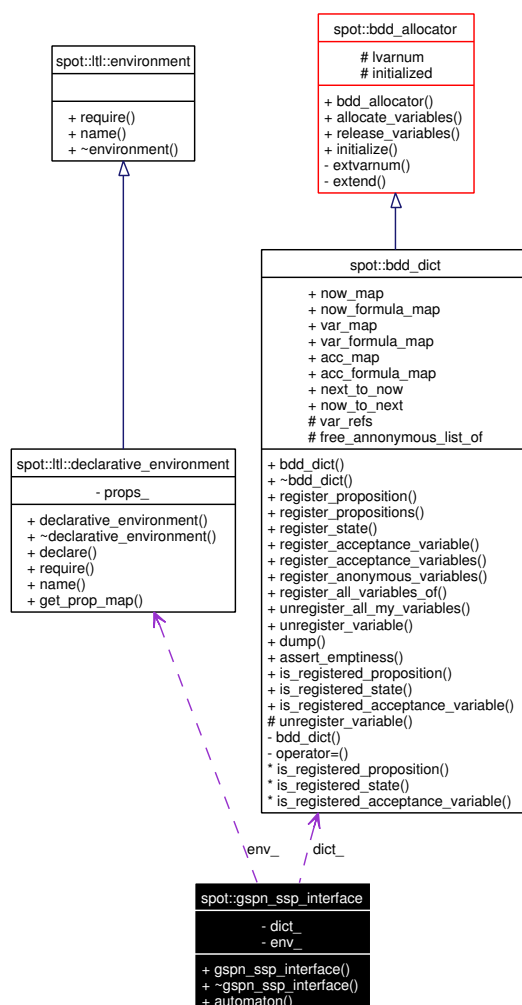
The documentation for this class was generated from the following file:

- gspn/gspn.hh

## 11.50 spot::gspn\_ssp\_interface Class Reference

```
#include <gspn/ssp.hh>
```

Collaboration diagram for spot::gspn\_ssp\_interface:



## Public Member Functions

- `gspn_ssp_interface` (int argc, char \*\*argv, `bdd_dict` \*dict, const `ltl::declarative_environment` &env, bool inclusion=false)
- `~gspn_ssp_interface` ()
- `tgba` \* `automaton` (const `tgba` \*operand) const

## Private Attributes

- `bdd_dict` \* `dict_`
- const `ltl::declarative_environment` & `env_`

### 11.50.1 Constructor & Destructor Documentation

**11.50.1.1** `spot::gspn_ssp_interface::gspn_ssp_interface` (int argc, char \*\* argv, `bdd_dict` \* dict, const `ltl::declarative_environment` & env, bool inclusion = false)

## 11.50.1.2 spot::gspn\_ssp\_interface::~gspn\_ssp\_interface ()

## 11.50.2 Member Function Documentation

## 11.50.2.1 tgba\* spot::gspn\_ssp\_interface::automaton (const tgba \* operand) const

## 11.50.3 Member Data Documentation

## 11.50.3.1 bdd\_dict\* spot::gspn\_ssp\_interface::dict\_ [private]

## 11.50.3.2 const ltl::declarative\_environment&amp; spot::gspn\_ssp\_interface::env\_ [private]

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

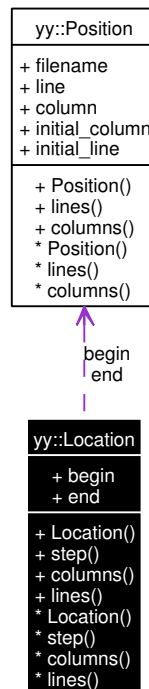
- [gspn/ssp.hh](#)

## 11.51 yy::Location Class Reference

Abstract a [Location](#).

```
#include <ltlparse/location.hh>
```

Collaboration diagram for yy::Location:



## Public Member Functions

**Ctor & dtor.**

- [Location](#) (void)  
*Construct a [Location](#).*

### Line and Column related manipulators

- void [step](#) (void)  
*Reset initial location to final location.*
- void [columns](#) (unsigned int count=1)  
*Extend the current location to the COUNT next columns.*
- void [lines](#) (unsigned int count=1)  
*Extend the current location to the COUNT next lines.*

### Public Attributes

- [Position begin](#)  
*Beginning of the located region.*
- [Position end](#)  
*End of the located region.*

#### 11.51.1 Detailed Description

Abstract a [Location](#).

#### 11.51.2 Constructor & Destructor Documentation

##### 11.51.2.1 yy::Location::Location (void) [inline]

Construct a [Location](#).

#### 11.51.3 Member Function Documentation

##### 11.51.3.1 void yy::Location::columns (unsigned int count = 1) [inline]

Extend the current location to the COUNT next columns.

##### 11.51.3.2 void yy::Location::lines (unsigned int count = 1) [inline]

Extend the current location to the COUNT next lines.

##### 11.51.3.3 void yy::Location::step (void) [inline]

Reset initial location to final location.

### 11.51.4 Member Data Documentation

#### 11.51.4.1 Position `yy::Location::begin`

Beginning of the located region.

#### 11.51.4.2 Position `yy::Location::end`

End of the located region.

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

- `Itlparse/location.hh`

## 11.52 spot::loopless\_modular\_mixed\_radix\_gray\_code Class Reference

Loopless modular mixed radix Gray code iteration.

```
#include <misc/modgray.hh>
```

### Public Member Functions

- `loopless_modular_mixed_radix_gray_code` (int n)
- virtual `~loopless_modular_mixed_radix_gray_code` ()

#### iteration over an element in a tuple

*The class does not know how to modify the elements of the tuple (Knuth's  $a_j$ 's). These changes are therefore abstracted using the `a_first()`, `a_next()`, and `a_last()` abstract functions. These need to be implemented in subclasses as appropriate.*

- virtual void `a_first` (int j)=0  
*Reset  $a_j$  to its initial value.*
- virtual void `a_next` (int j)=0  
*Advance  $a_j$  to its next value.*
- virtual bool `a_last` (int j) const =0  
*Whether  $a_j$  is on its last value.*

#### iteration over all the tuples

- void `first` ()  
*Reset the iteration to the first tuple.*
- bool `last` () const  
*Whether this the last tuple.*
- bool `done` () const  
*Whether all tuple have been explored.*
- int `next` ()  
*Update one item of the tuple and return its position.*

## Protected Attributes

- int `n_`
- bool `done_`
- int \* `a_`
- int \* `f_`
- int \* `m_`
- int \* `s_`
- int \* `non_one_radixes_`

### 11.52.1 Detailed Description

Loopless modular mixed radix Gray code iteration.

This class is based on the loopless modular mixed radix gray code algorithm described in exercise 77 of "The Art of Computer Programming", Pre-Fascicle 2A (Draft of section 7.2.1.1: generating all n-tuples) by Donald E. Knuth.

The idea is to enumerate the set of all n-tuples  $(a_0, a_1, \dots, a_{n-1})$  where each  $a_j$  range over a distinct set (this is the *mixed radix* part), so that only one  $a_j$  changes between two successive tuples of the iteration (that is the *Gray code* part), and that this changes occurs always in the same direction, cycling over the set  $a_j$  must cover (i.e., *modular*). The algorithm is *loopless* in that computing the next tuple done without any loop, i.e., in constant time.

This class does not need to know the type of the  $a_j$ , it will handle them indirectly through three methods: `a_first()`, `a_next()`, and `a_last()`. These methods need to be implemented in a subclass for the particular type of  $a_j$  at hand.

The class itself offers four functions to control the iteration over the set of all the  $(a_0, a_1, \dots, a_{n-1})$  tuples: `first()`, `next()`, `last()`, and `done()`. These functions are usually used as follows:

```
for (g.first(); !g.done(); g.next())
    use the tuple
```

How to use the tuple of course depends on the way it as been stored in the subclass.

Finally, let's mention two differences between this algorithm and the one in Knuth's book. This version of the algorithm does not need to know the radixes (i.e., the size of set of each  $a_j$ ) beforehand: it will discover them on-the-fly when `a_last(j)` first return true. It will also work with  $a_j$  that cannot be changed. (This is achieved by reindexing the elements through `non_one_radixes_`, to consider only the elements with a non-singleton range.)

### 11.52.2 Constructor & Destructor Documentation

#### 11.52.2.1 spot::loopless\_modular\_mixed\_radix\_gray\_code::loopless\_modular\_mixed\_radix\_gray\_code (int *n*)

Constructor.

#### Parameters:

- n* The size of the tuples to enumerate.

#### 11.52.2.2 virtual spot::loopless\_modular\_mixed\_radix\_gray\_code::~loopless\_modular\_mixed\_radix\_gray\_code () [virtual]



### 11.52.3 Member Function Documentation

**11.52.3.1** `virtual void spot::loopless_modular_mixed_radix_gray_code::a_first (int j) [pure virtual]`

Reset  $a_j$  to its initial value.

**11.52.3.2** `virtual bool spot::loopless_modular_mixed_radix_gray_code::a_last (int j) const [pure virtual]`

Whether  $a_j$  is on its last value.

**11.52.3.3** `virtual void spot::loopless_modular_mixed_radix_gray_code::a_next (int j) [pure virtual]`

Advance  $a_j$  to its next value.

This will never be called if `a_last(j)` is true.

**11.52.3.4** `bool spot::loopless_modular_mixed_radix_gray_code::done () const [inline]`

Whether all tuple have been explored.

**11.52.3.5** `void spot::loopless_modular_mixed_radix_gray_code::first ()`

Reset the iteration to the first tuple.

This must be called before calling any of `next()`, `last()`, or `done()`.

**11.52.3.6** `bool spot::loopless_modular_mixed_radix_gray_code::last () const [inline]`

Whether this the last tuple.

At this point it is still OK to call `next()`, and then `done()` will become true.

**11.52.3.7** `int spot::loopless_modular_mixed_radix_gray_code::next ()`

Update one item of the tuple and return its position.

`next()` should never be called if `done()` is true. If it is called on the last tuple (i.e., `last()` is true), it will return -1. Otherwise it will update one  $a_j$  of the tuple through one the  $a_j$  handling functions, and return  $j$ .

### 11.52.4 Member Data Documentation

**11.52.4.1** `int* spot::loopless_modular_mixed_radix_gray_code::a_ [protected]`

**11.52.4.2** `bool spot::loopless_modular_mixed_radix_gray_code::done_ [protected]`

**11.52.4.3** `int* spot::loopless_modular_mixed_radix_gray_code::f_ [protected]`

**11.52.4.4** `int* spot::loopless_modular_mixed_radix_gray_code::m_ [protected]`

11.52.4.5 `int` `spot::loopless_modular_mixed_radix_gray_code::n_` [protected]

11.52.4.6 `int*` `spot::loopless_modular_mixed_radix_gray_code::non_one_radixes_` [protected]

11.52.4.7 `int*` `spot::loopless_modular_mixed_radix_gray_code::s_` [protected]

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

- `misc/modgray.hh`

## 11.53 `spot::minato_isop` Class Reference

Generate an irredundant sum-of-products (ISOP) form of a BDD function.

```
#include <misc/minato.hh>
```

### Public Member Functions

- `minato_isop` (bdd input)  
*Constructor.*
  - *input* The BDD function to translate in ISOP.
- `minato_isop` (bdd input, bdd vars)  
*Constructor.*
  - *input* The BDD function to translate in ISOP.
  - *vars* The set of BDD variables to factorize in input.
- `bdd next ()`  
*Compute the next sum term of the ISOP form. Return `bddfalse` when all terms have been output.*

### Private Attributes

- `std::stack< local_vars > todo_`
- `std::stack< bdd > cube_`
- `bdd ret_`

### Classes

- `struct local_vars`  
*Internal variables for `minato_isop`.*

### 11.53.1 Detailed Description

Generate an irredundant sum-of-products (ISOP) form of a BDD function.

This algorithm implements a derecursed version the Minato-Morreale algorithm presented in the following paper.

```
@InProceedings{ minato.92.sasimi,
  author      = {Shin-ichi Minato},
  title       = {Fast Generation of Irredundant Sum-of-Products Forms
                from Binary Decision Diagrams},
  booktitle   = {Proceedings of the third Synthesis and Simulation
                and Meeting International Interchange workshop
                (SASIMI'92)},
  pages       = {64--73},
  year        = {1992},
  address     = {Kobe, Japan},
  month       = {April}
}
```

### 11.53.2 Constructor & Destructor Documentation

#### 11.53.2.1 spot::minato\_isop::minato\_isop (bdd *input*)

Constructor.

- *input* The BDD function to translate in ISOP.

#### 11.53.2.2 spot::minato\_isop::minato\_isop (bdd *input*, bdd *vars*)

Constructor.

- *input* The BDD function to translate in ISOP.
- *vars* The set of BDD variables to factorize in *input*.

### 11.53.3 Member Function Documentation

#### 11.53.3.1 bdd spot::minato\_isop::next ()

Compute the next sum term of the ISOP form. Return `bddfalse` when all terms have been output.

### 11.53.4 Member Data Documentation

#### 11.53.4.1 std::stack<bdd> spot::minato\_isop::cube\_ [private]

#### 11.53.4.2 bdd spot::minato\_isop::ret\_ [private]

#### 11.53.4.3 std::stack<local\_vars> spot::minato\_isop::todo\_ [private]

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

- [misc/minato.hh](#)

## 11.54 spot::minato\_isop::local\_vars Struct Reference

Internal variables for [minato\\_isop](#).

### Public Types

- enum { [FirstStep](#), [SecondStep](#), [ThirdStep](#), [FourthStep](#) }

### Public Member Functions

- [local\\_vars](#) (bdd [f\\_min](#), bdd [f\\_max](#), bdd [vars](#))

### Public Attributes

- bdd [f\\_min](#)
- bdd [f\\_max](#)
- enum spot::minato\_isop::local\_vars:: { ... } [step](#)
- bdd [vars](#)
- bdd [v1](#)
- bdd [f0\\_min](#)
- bdd [f0\\_max](#)
- bdd [f1\\_min](#)
- bdd [f1\\_max](#)
- bdd [g0](#)
- bdd [g1](#)

#### 11.54.1 Detailed Description

Internal variables for [minato\\_isop](#).

#### 11.54.2 Member Enumeration Documentation

##### 11.54.2.1 anonymous enum

Enumeration values:

*FirstStep*

*SecondStep*

*ThirdStep*

*FourthStep*

#### 11.54.3 Constructor & Destructor Documentation

11.54.3.1 [spot::minato\\_isop::local\\_vars::local\\_vars](#) (bdd [f\\_min](#), bdd [f\\_max](#), bdd [vars](#)) `[inline]`

#### 11.54.4 Member Data Documentation

11.54.4.1 bdd [spot::minato\\_isop::local\\_vars::f0\\_max](#)

11.54.4.2 bdd [spot::minato\\_isop::local\\_vars::f0\\_min](#)

11.54.4.3 bdd [spot::minato\\_isop::local\\_vars::f1\\_max](#)

11.54.4.4 bdd [spot::minato\\_isop::local\\_vars::f1\\_min](#)

11.54.4.5 bdd [spot::minato\\_isop::local\\_vars::f\\_max](#)

11.54.4.6 bdd [spot::minato\\_isop::local\\_vars::f\\_min](#)

11.54.4.7 bdd [spot::minato\\_isop::local\\_vars::g0](#)

11.54.4.8 bdd [spot::minato\\_isop::local\\_vars::g1](#)

11.54.4.9 enum { ... } [spot::minato\\_isop::local\\_vars::step](#)

11.54.4.10 bdd [spot::minato\\_isop::local\\_vars::v1](#)

11.54.4.11 bdd [spot::minato\\_isop::local\\_vars::vars](#)

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

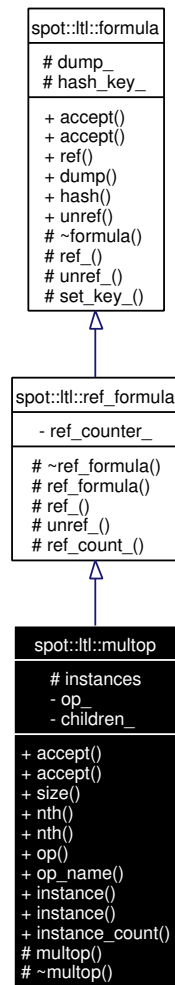
- [misc/minato.hh](#)

## 11.55 spot::ltl::multop Class Reference

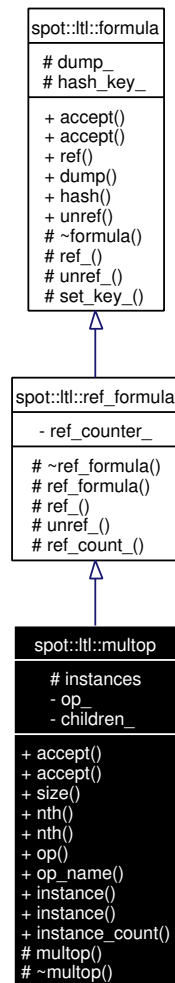
Multi-operand operators.

```
#include <ltlast/multop.hh>
```

Inheritance diagram for spot::ltl::multop:



Collaboration diagram for `spot::ltl::multop`:



## Public Types

- typedef `std::vector< formula * >` `vec`  
*List of formulae.*
- enum `type` { `Or`, `And` }

## Public Member Functions

- virtual void `accept` (`visitor &v`)  
*Entry point for `vspot::ltl::visitor` instances.*
- virtual void `accept` (`const_visitor &v`) `const`  
*Entry point for `vspot::ltl::const_visitor` instances.*
- unsigned `size` () `const`  
*Get the number of children.*

- const formula \* nth (unsigned n) const  
*Get the nth children.*
- formula \* nth (unsigned n)  
*Get the nth children.*
- type op () const  
*Get the type of this operator.*
- const char \* op\_name () const  
*Get the type of this operator, as a string.*
- formula \* ref ()  
*clone this node*
- const std::string & dump () const  
*Return a canonic representation of the formula.*
- const size\_t hash () const  
*Return a hash\_key for the formula.*

### Static Public Member Functions

- static formula \* instance (type op, formula \*first, formula \*second)  
*Build a spot::ltl::multop with two children.*
- static formula \* instance (type op, vec \*v)  
*Build a spot::ltl::multop with many children.*
- static unsigned instance\_count ()  
*Number of instantiated multi-operand operators. For debugging.*
- static void unref (formula \*f)  
*release this node*

### Protected Types

- typedef std::pair< type, vec \* > pair
- typedef std::map< pair, formula \*, paircmp > map

### Protected Member Functions

- multop (type op, vec \*v)
- virtual ~multop ()
- void ref\_ ()  
*increment reference counter if any*



- bool `unref_()`  
*decrement reference counter if any, return true when the instance must be deleted (usually when the counter hits 0).*
- unsigned `ref_count_()`  
*Number of references to this formula.*
- void `set_key_()`  
*Compute key\_ from dump\_.*

### Protected Attributes

- std::string `dump_`  
*The canonic representation of the formula.*
- size\_t `hash_key_`  
*The hash key of this formula.*

### Static Protected Attributes

- static `map instances`

### Private Attributes

- `type op_`
- `vec * children_`

### Classes

- struct `paircmp`  
*Comparison functor used internally by `ltl::multop`.*

## 11.55.1 Detailed Description

Multi-operand operators.

These operators are considered commutative and associative.

## 11.55.2 Member Typedef Documentation

**11.55.2.1** `typedef std::map<pair, formula*, paircmp> spot::ltl::multop::map` [protected]

**11.55.2.2** `typedef std::pair<type, vec*> spot::ltl::multop::pair` [protected]

**11.55.2.3** typedef std::vector<formula\*> spot::ltl::multop::vec

List of formulae.

**11.55.3** Member Enumeration Documentation**11.55.3.1** enum spot::ltl::multop::type

Enumeration values:

*Or*

*And*

**11.55.4** Constructor & Destructor Documentation**11.55.4.1** spot::ltl::multop::multop (type op, vec \* v) [protected]**11.55.4.2** virtual spot::ltl::multop::~~multop () [protected, virtual]**11.55.5** Member Function Documentation**11.55.5.1** virtual void spot::ltl::multop::accept (const\_visitor & v) const [virtual]

Entry point for vspot::ltl::const\_visitor instances.

Implements spot::ltl::formula.

**11.55.5.2** virtual void spot::ltl::multop::accept (visitor & v) [virtual]

Entry point for vspot::ltl::visitor instances.

Implements spot::ltl::formula.

**11.55.5.3** const std::string& spot::ltl::formula::dump () const [inherited]

Return a canonic representation of the formula.

**11.55.5.4** const size\_t spot::ltl::formula::hash () const [inline, inherited]

Return a hash\_key for the formula.

**11.55.5.5** static formula\* spot::ltl::multop::instance (type op, vec \* v) [static]

Build a spot::ltl::multop with many children.

Same as the other [instance\(\)](#) function, but take a vector of formula in argument. This vector is acquired by the spot::ltl::multop class, the caller should allocate it with new, but not use it (especially not destroy it) after it has been passed to spot::ltl::multop.

This functions can perform slight optimizations and may not return an [ltl::multop](#) objects. For instance if the vector contain only one unique element, this this formula will be returned as-is.

**11.55.5.6** `static formula* spot::ltl::multop::instance (type op, formula * first, formula * second)` `[static]`

Build a `spot::ltl::multop` with two children.

If one of the children itself is a `spot::ltl::multop` with the same type, it will be merged. I.e., children if that child will be added, and that child itself will be destroyed. This allows incremental building of n-ary `ltl::multop`.

This functions can perform slight optimizations and may not return an `ltl::multop` objects. For instance if `first` and `second` are equal, that formula is returned as-is.

**11.55.5.7** `static unsigned spot::ltl::multop::instance_count ()` `[static]`

Number of instantiated multi-operand operators. For debugging.

**11.55.5.8** `formula* spot::ltl::multop::nth (unsigned n)`

Get the `n`th children.

Starting with `n = 0`.

**11.55.5.9** `const formula* spot::ltl::multop::nth (unsigned n) const`

Get the `n`th children.

Starting with `n = 0`.

**11.55.5.10** `type spot::ltl::multop::op () const`

Get the type of this operator.

**11.55.5.11** `const char* spot::ltl::multop::op_name () const`

Get the type of this operator, as a string.

**11.55.5.12** `formula* spot::ltl::formula::ref ()` `[inherited]`

clone this node

This increments the reference counter of this node (if one is used). You should almost never use this method directly as it doesn't touch the children. If you want to clone a whole formula, use `spot::ltl::clone()` instead.

**11.55.5.13** `void spot::ltl::ref_formula::ref_ ()` `[protected, virtual, inherited]`

increment reference counter if any

Reimplemented from `spot::ltl::formula`.

**11.55.5.14** `unsigned spot::ltl::ref_formula::ref_count_ ()` `[protected, inherited]`

Number of references to this formula.

**11.55.5.15** `void spot::ltl::formula::set_key_()` [protected, inherited]

Compute `key_` from `dump_`.

Should be called once in each object, after `dump_` has been set.

**11.55.5.16** `unsigned spot::ltl::multop::size() const`

Get the number of children.

**11.55.5.17** `static void spot::ltl::formula::unref(formula *f)` [static, inherited]

release this node

This decrements the reference counter of this node (if one is used) and can free the object. You should almost never use this method directly as it doesn't touch the children. If you want to release a whole formula, use `spot::ltl::destroy()` instead.

**11.55.5.18** `bool spot::ltl::ref_formula::unref_()` [protected, virtual, inherited]

decrement reference counter if any, return true when the instance must be deleted (usually when the counter hits 0).

Reimplemented from `spot::ltl::formula`.

**11.55.6** Member Data Documentation**11.55.6.1** `vec* spot::ltl::multop::children_` [private]**11.55.6.2** `std::string spot::ltl::formula::dump_` [protected, inherited]

The canonic representation of the formula.

**11.55.6.3** `size_t spot::ltl::formula::hash_key_` [protected, inherited]

The hash key of this formula.

Initialized by `set_key_()`.

**11.55.6.4** `map spot::ltl::multop::instances` [static, protected]**11.55.6.5** `type spot::ltl::multop::op_` [private]

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

- `ltlast/multop.hh`

**11.56** `spot::ltl::multop::pairemp` Struct Reference

Comparison functor used internally by `ltl::multop`.

```
#include <ltlast/multop.hh>
```

## Public Member Functions

- bool [operator\(\)](#) (const [pair](#) &p1, const [pair](#) &p2) const

### 11.56.1 Detailed Description

Comparison functor used internally by [ltl::multop](#).

### 11.56.2 Member Function Documentation

**11.56.2.1** bool [spot::ltl::multop::paircmp::operator\(\)](#) (const [pair](#) & *p1*, const [pair](#) & *p2*) const  
[inline]

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

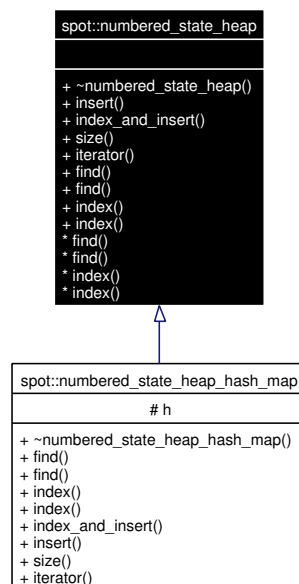
- [ltlast/multop.hh](#)

## 11.57 spot::numbered\_state\_heap Class Reference

Keep track of a large quantity of indexed states.

```
#include <tgbalgorithms/gtec/nsheap.hh>
```

Inheritance diagram for [spot::numbered\\_state\\_heap](#):



## Public Types

- typedef std::pair< const [state](#) \*, int \* > [state\\_index\\_p](#)
- typedef std::pair< const [state](#) \*, int > [state\\_index](#)

**Public Member Functions**

- virtual `~numbered_state_heap` ()
- virtual void `insert` (const `state` \*s, int index)=0  
*Add a new state s with index index.*
- virtual int & `index_and_insert` (const `state` \*&s)=0  
*Get the index of a state, and insert that state if it is missing.*
- virtual int `size` () const =0  
*The number of stored states.*
- virtual `numbered_state_heap_const_iterator` \* `iterator` () const =0  
*Return an iterator on the states/indexes pairs.*
- virtual `state_index` `find` (const `state` \*s) const =0  
*Is state in the heap?*
- virtual `state_index_p` `find` (const `state` \*s)=0
- virtual `state_index` `index` (const `state` \*s) const =0  
*Return the index of an existing state.*
- virtual `state_index_p` `index` (const `state` \*s)=0

**11.57.1 Detailed Description**

Keep track of a large quantity of indexed states.

**11.57.2 Member Typedef Documentation**

**11.57.2.1** `typedef std::pair<const state*, int> spot::numbered_state_heap::state_index`

**11.57.2.2** `typedef std::pair<const state*, int*> spot::numbered_state_heap::state_index_p`

**11.57.3 Constructor & Destructor Documentation**

**11.57.3.1** virtual `spot::numbered_state_heap::~numbered_state_heap` () [inline, virtual]

**11.57.4 Member Function Documentation**

**11.57.4.1** virtual `state_index_p` `spot::numbered_state_heap::find` (const `state` \* s) [pure virtual]

Implemented in `spot::numbered_state_heap_hash_map`.

**11.57.4.2** `virtual state_index spot::numbered_state_heap::find (const state * s) const` [pure virtual]

Is state in the heap?

Returns a pair (0,0) if *s* is not in the heap. or a pair (*p*, *i*) if there is a clone *p* of *s* *i* in the heap with index. If *s* is in the heap and is different from *p* it will be freed.

These functions are called by the algorithm to check whether a successor is a new state to explore or an already visited state.

These functions can be redefined to search for more than an equal match. For example we could redefine it to check state inclusion.

Implemented in `spot::numbered_state_heap_hash_map`.

**11.57.4.3** `virtual state_index_p spot::numbered_state_heap::index (const state * s)` [pure virtual]

Implemented in `spot::numbered_state_heap_hash_map`.

**11.57.4.4** `virtual state_index spot::numbered_state_heap::index (const state * s) const` [pure virtual]

Return the index of an existing state.

This is mostly similar to `find()`, except it will be called for state which we know are already in the heap, or for state which may not be in the heap but for which it is always OK to do equality checks.

Implemented in `spot::numbered_state_heap_hash_map`.

**11.57.4.5** `virtual int& spot::numbered_state_heap::index_and_insert (const state *& s)` [pure virtual]

Get the index of a state, and insert that state if it is missing.

If a clone of *s* is already in the hash table, *s* will be deleted and replaced by the address of the clone used.

Implemented in `spot::numbered_state_heap_hash_map`.

**11.57.4.6** `virtual void spot::numbered_state_heap::insert (const state * s, int index)` [pure virtual]

Add a new state *s* with index *index*.

Implemented in `spot::numbered_state_heap_hash_map`.

**11.57.4.7** `virtual numbered_state_heap_const_iterator* spot::numbered_state_heap::iterator () const` [pure virtual]

Return an iterator on the states/indexes pairs.

Implemented in `spot::numbered_state_heap_hash_map`.

**11.57.4.8** `virtual int spot::numbered_state_heap::size () const` [pure virtual]

The number of stored states.

Implemented in `spot::numbered_state_heap_hash_map`.

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

- [tgbalgorithms/gtec/nsheap.hh](#)

## 11.58 spot::numbered\_state\_heap\_const\_iterator Class Reference

Iterator on [numbered\\_state\\_heap](#) objects.

```
#include <tgbalgorithms/gtec/nsheap.hh>
```

### Public Member Functions

- virtual [~numbered\\_state\\_heap\\_const\\_iterator](#) ()
- virtual void [first](#) ()=0  
*Iteration.*
- virtual void [next](#) ()=0
- virtual bool [done](#) () const =0
- virtual const [state](#) \* [get\\_state](#) () const =0  
*Inspection.*
- virtual int [get\\_index](#) () const =0

#### 11.58.1 Detailed Description

Iterator on [numbered\\_state\\_heap](#) objects.

#### 11.58.2 Constructor & Destructor Documentation

**11.58.2.1** virtual [spot::numbered\\_state\\_heap\\_const\\_iterator::~numbered\\_state\\_heap\\_const\\_iterator](#) () [inline, virtual]

#### 11.58.3 Member Function Documentation

**11.58.3.1** virtual bool [spot::numbered\\_state\\_heap\\_const\\_iterator::done](#) () const [pure virtual]

**11.58.3.2** virtual void [spot::numbered\\_state\\_heap\\_const\\_iterator::first](#) () [pure virtual]

Iteration.

**11.58.3.3** virtual int [spot::numbered\\_state\\_heap\\_const\\_iterator::get\\_index](#) () const [pure virtual]

**11.58.3.4** virtual const [state](#)\* [spot::numbered\\_state\\_heap\\_const\\_iterator::get\\_state](#) () const [pure virtual]

Inspection.



### 11.58.3.5 virtual void spot::numbered\_state\_heap\_const\_iterator::next () [pure virtual]

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

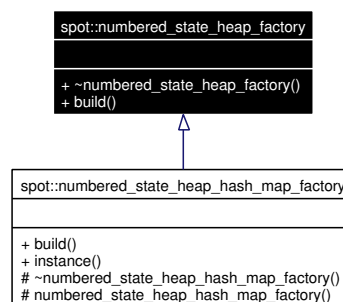
- [tgbalgorithms/gtec/nsheap.hh](#)

## 11.59 spot::numbered\_state\_heap\_factory Class Reference

Abstract factory for [numbered\\_state\\_heap](#).

```
#include <tgbalgorithms/gtec/nsheap.hh>
```

Inheritance diagram for spot::numbered\_state\_heap\_factory:



### Public Member Functions

- virtual `~numbered_state_heap_factory ()`
- virtual `numbered_state_heap * build () const =0`

### 11.59.1 Detailed Description

Abstract factory for [numbered\\_state\\_heap](#).

### 11.59.2 Constructor & Destructor Documentation

**11.59.2.1** virtual `spot::numbered_state_heap_factory::~~numbered_state_heap_factory ()`  
 [inline, virtual]

### 11.59.3 Member Function Documentation

**11.59.3.1** virtual `numbered_state_heap* spot::numbered_state_heap_factory::build () const`  
 [pure virtual]

Implemented in [spot::numbered\\_state\\_heap\\_hash\\_map\\_factory](#).

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

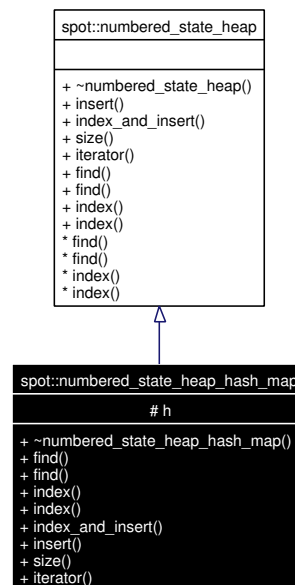
- [tgbalgorithms/gtec/nsheap.hh](#)

## 11.60 spot::numbered\_state\_heap\_hash\_map Class Reference

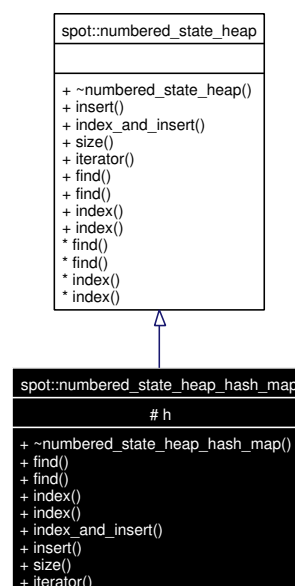
A straightforward implementation of [numbered\\_state\\_heap](#) with a hash map.

```
#include <tgbalgorithms/gtec/nsheap.hh>
```

Inheritance diagram for spot::numbered\_state\_heap\_hash\_map:



Collaboration diagram for spot::numbered\_state\_heap\_hash\_map:



## Public Types

- typedef Sgi::hash\_map< const [state](#) \*, int, [state\\_ptr\\_hash](#), [state\\_ptr\\_equal](#) > [hash\\_type](#)
- typedef std::pair< const [state](#) \*, int \* > [state\\_index\\_p](#)
- typedef std::pair< const [state](#) \*, int > [state\\_index](#)

## Public Member Functions

- virtual [~numbered\\_state\\_heap\\_hash\\_map](#) ()
- virtual [state\\_index](#) [find](#) (const [state](#) \*s) const  
*Is state in the heap?*
- virtual [state\\_index\\_p](#) [find](#) (const [state](#) \*s)
- virtual [state\\_index](#) [index](#) (const [state](#) \*s) const  
*Return the index of an existing state.*
- virtual [state\\_index\\_p](#) [index](#) (const [state](#) \*s)
- virtual int & [index\\_and\\_insert](#) (const [state](#) \*&s)  
*Get the index of a state, and insert that state if it is missing.*
- virtual void [insert](#) (const [state](#) \*s, int index)  
*Add a new state s with index index.*
- virtual int [size](#) () const  
*The number of stored states.*
- virtual [numbered\\_state\\_heap\\_const\\_iterator](#) \* [iterator](#) () const  
*Return an iterator on the states/indexes pairs.*

## Protected Attributes

- [hash\\_type](#) h  
*Map of visited states.*

### 11.60.1 Detailed Description

A straightforward implementation of [numbered\\_state\\_heap](#) with a hash map.

### 11.60.2 Member Typedef Documentation

**11.60.2.1** typedef Sgi::hash\_map<const [state](#)\*, int, [state\\_ptr\\_hash](#), [state\\_ptr\\_equal](#)>  
[spot::numbered\\_state\\_heap\\_hash\\_map::hash\\_type](#)

**11.60.2.2** typedef std::pair<const [state](#)\*, int> [spot::numbered\\_state\\_heap::state\\_index](#)  
[inherited]

**11.60.2.3** `typedef std::pair<const state*, int*> spot::numbered_state_heap::state_index_p`  
[inherited]

### 11.60.3 Constructor & Destructor Documentation

**11.60.3.1** `virtual spot::numbered_state_heap_hash_map::~~numbered_state_heap_hash_map ()`  
[virtual]

### 11.60.4 Member Function Documentation

**11.60.4.1** `virtual state_index_p spot::numbered_state_heap_hash_map::find (const state * s)`  
[virtual]

Implements [spot::numbered\\_state\\_heap](#).

**11.60.4.2** `virtual state_index spot::numbered_state_heap_hash_map::find (const state * s) const`  
[virtual]

Is state in the heap?

Returns a pair (0,0) if *s* is not in the heap. or a pair (p, i) if there is a clone *p* of *s* *i* in the heap with index. If *s* is in the heap and is different from *p* it will be freed.

These functions are called by the algorithm to check whether a successor is a new state to explore or an already visited state.

These functions can be redefined to search for more than an equal match. For example we could redefine it to check state inclusion.

Implements [spot::numbered\\_state\\_heap](#).

**11.60.4.3** `virtual state_index_p spot::numbered_state_heap_hash_map::index (const state * s)`  
[virtual]

Implements [spot::numbered\\_state\\_heap](#).

**11.60.4.4** `virtual state_index spot::numbered_state_heap_hash_map::index (const state * s) const`  
[virtual]

Return the index of an existing state.

This is mostly similar to [find\(\)](#), except it will be called for state which we know are already in the heap, or for state which may not be in the heap but for which it is always OK to do equality checks.

Implements [spot::numbered\\_state\\_heap](#).

**11.60.4.5** `virtual int& spot::numbered_state_heap_hash_map::index_and_insert (const state *& s)`  
[virtual]

Get the index of a state, and insert that state if it is missing.

If a clone of *s* is already in the hash table, *s* will be deleted and replaced by the address of the clone used.

Implements [spot::numbered\\_state\\_heap](#).

**11.60.4.6** `virtual void spot::numbered_state_heap_hash_map::insert (const state * s, int index)` `[virtual]`

Add a new state *s* with index *index*.

Implements [spot::numbered\\_state\\_heap](#).

**11.60.4.7** `virtual numbered\_state\_heap\_const\_iterator* spot::numbered_state_heap_hash_map::iterator () const` `[virtual]`

Return an iterator on the states/indexes pairs.

Implements [spot::numbered\\_state\\_heap](#).

**11.60.4.8** `virtual int spot::numbered_state_heap_hash_map::size () const` `[virtual]`

The number of stored states.

Implements [spot::numbered\\_state\\_heap](#).

## 11.60.5 Member Data Documentation

**11.60.5.1** `hash\_type spot::numbered_state_heap_hash_map::h` `[protected]`

Map of visited states.

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

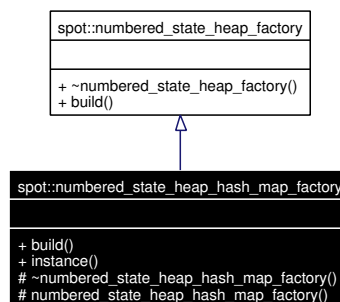
- [tgbaalgos/gtec/nsheap.hh](#)

## 11.61 spot::numbered\_state\_heap\_hash\_map\_factory Class Reference

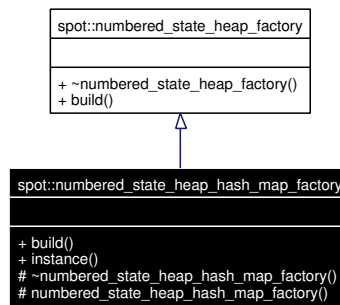
Factory for [numbered\\_state\\_heap\\_hash\\_map](#).

```
#include <tgbaalgos/gtec/nsheap.hh>
```

Inheritance diagram for `spot::numbered_state_heap_hash_map_factory`:



Collaboration diagram for `spot::numbered_state_heap_hash_map_factory`:



### Public Member Functions

- virtual [numbered\\_state\\_heap\\_hash\\_map](#) \* `build` () const

### Static Public Member Functions

- static const [numbered\\_state\\_heap\\_hash\\_map\\_factory](#) \* `instance` ()  
*Get the unique instance of this class.*

### Protected Member Functions

- virtual `~numbered_state_heap_hash_map_factory` ()
- `numbered_state_heap_hash_map_factory` ()

#### 11.61.1 Detailed Description

Factory for [numbered\\_state\\_heap\\_hash\\_map](#).

This class is a singleton. Retrieve the instance using [instance\(\)](#).

#### 11.61.2 Constructor & Destructor Documentation

**11.61.2.1** virtual `spot::numbered_state_heap_hash_map_factory::~~numbered_state_heap_hash_map_factory` () [inline, protected, virtual]

**11.61.2.2** `spot::numbered_state_heap_hash_map_factory::numbered_state_heap_hash_map_factory` () [protected]

#### 11.61.3 Member Function Documentation

**11.61.3.1** virtual `numbered_state_heap_hash_map`\* `spot::numbered_state_heap_hash_map_factory::build` () const [virtual]

Implements [spot::numbered\\_state\\_heap\\_factory](#).

### 11.61.3.2 static const [numbered\\_state\\_heap\\_hash\\_map\\_factory\\*](#) spot::numbered\_state\_heap\_hash\_map\_factory::instance() [static]

Get the unique instance of this class.

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

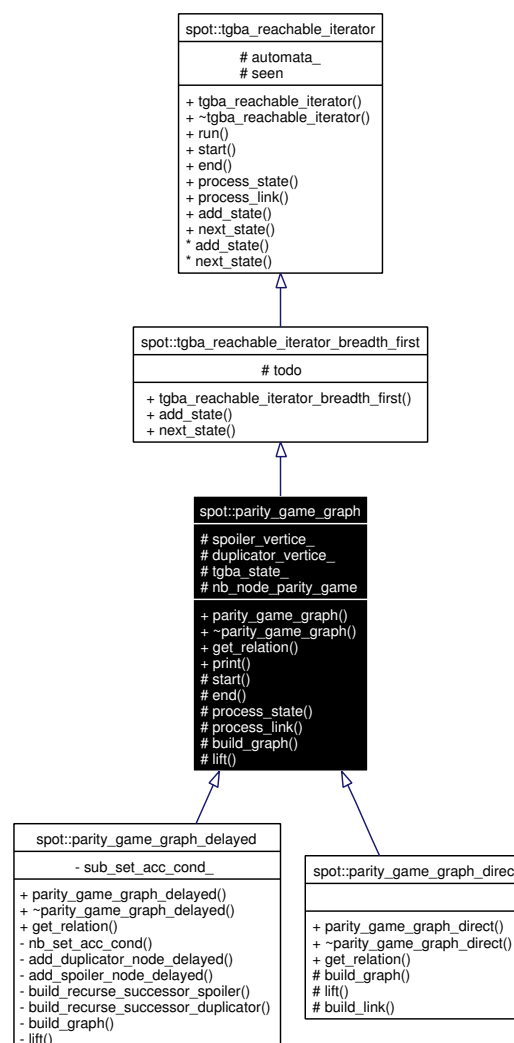
- [tgbaalgos/gtec/nsheap.hh](#)

## 11.62 spot::parity\_game\_graph Class Reference

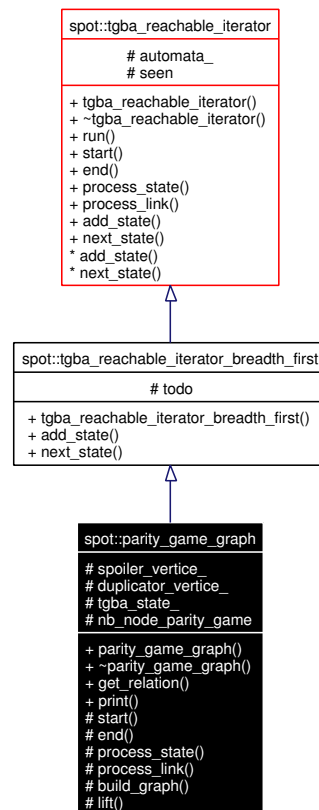
Parity game graph which compute a simulation relation.

```
#include <tgbaalgos/reductgba_sim.hh>
```

Inheritance diagram for spot::parity\_game\_graph:



Collaboration diagram for spot::parity\_game\_graph:



## Public Member Functions

- [parity\\_game\\_graph](#) (const [tgba](#) \*a)
- virtual [~parity\\_game\\_graph](#) ()
- virtual [simulation\\_relation](#) \* [get\\_relation](#) ()=0
- void [print](#) (std::ostream &os)
- virtual void [add\\_state](#) (const [state](#) \*s)
- virtual const [state](#) \* [next\\_state](#) ()  
*Called by [run\(\)](#) to obtain the.*
- void [run](#) ()  
*Iterate over all reachable states of a [spot::tgba](#).*
- virtual void [process\\_link](#) (const [state](#) \*in\_s, int in, const [state](#) \*out\_s, int out, const [tgba\\_succ\\_iterator](#) \*si)

## Protected Types

- typedef Sgi::hash\_map< const [state](#) \*, int, [state\\_ptr\\_hash](#), [state\\_ptr\\_equal](#) > [seen\\_map](#)

## Protected Member Functions

- void [start](#) ()



Called by [run\(\)](#) before starting its iteration.

- void [end](#) ()

Called by [run\(\)](#) once all states have been explored.

- void [process\\_state](#) (const [state](#) \*s, int n, [tgba\\_succ\\_iterator](#) \*si)
- void [process\\_link](#) (int in, int out, const [tgba\\_succ\\_iterator](#) \*si)
- virtual void [build\\_graph](#) ()=0

Compute each node of the graph.

- virtual void [lift](#) ()=0

Remove edge from spoiler to duplicator that make duplicator loose. Spoiler node whose still have some link, reveal a direct simulation relation.

## Protected Attributes

- [sn\\_v](#) spoiler\_vertice\_
- [dn\\_v](#) duplicator\_vertice\_
- [s\\_v](#) tgba\_state\_
- int [nb\\_node\\_parity\\_game](#)
- std::deque< const [state](#) \* > [todo](#)

A queue of states yet to explore.

- const [tgba](#) \* [automata\\_](#)
- The [spot::tgba](#) to explore.

- [seen\\_map](#) seen
- States already seen.

### 11.62.1 Detailed Description

Parity game graph which compute a simulation relation.

### 11.62.2 Member Typedef Documentation

**11.62.2.1** typedef Sgi::hash\_map<const [state](#)\*, int, [state\\_ptr\\_hash](#), [state\\_ptr\\_equal](#)> [spot::tgba\\_reachable\\_iterator::seen\\_map](#) [protected, inherited]

Reimplemented in [spot::tgba\\_reduc](#).

### 11.62.3 Constructor & Destructor Documentation

**11.62.3.1** [spot::parity\\_game\\_graph::parity\\_game\\_graph](#) (const [tgba](#) \* a)

**11.62.3.2** virtual [spot::parity\\_game\\_graph::~~parity\\_game\\_graph](#) () [virtual]

### 11.62.4 Member Function Documentation

**11.62.4.1** `virtual void spot::tgba_reachable_iterator_breadth_first::add_state (const state * s)` [virtual, inherited]

Implements [spot::tgba\\_reachable\\_iterator](#).

**11.62.4.2** `virtual void spot::parity_game_graph::build_graph ()` [protected, pure virtual]

Compute each node of the graph.

Implemented in [spot::parity\\_game\\_graph\\_direct](#), and [spot::parity\\_game\\_graph\\_delayed](#).

**11.62.4.3** `void spot::parity_game_graph::end ()` [protected, virtual]

Called by [run\(\)](#) once all states have been explored.

Reimplemented from [spot::tgba\\_reachable\\_iterator](#).

**11.62.4.4** `virtual simulation\_relation* spot::parity_game_graph::get_relation ()` [pure virtual]

Implemented in [spot::parity\\_game\\_graph\\_direct](#), and [spot::parity\\_game\\_graph\\_delayed](#).

**11.62.4.5** `virtual void spot::parity_game_graph::lift ()` [protected, pure virtual]

Remove edge from spoiler to duplicator that make duplicator loose. Spoiler node whose still have some link, reveal a direct simulation relation.

Implemented in [spot::parity\\_game\\_graph\\_direct](#), and [spot::parity\\_game\\_graph\\_delayed](#).

**11.62.4.6** `virtual const state* spot::tgba_reachable_iterator_breadth_first::next_state ()` [virtual, inherited]

Called by [run\(\)](#) to obtain the.

Implements [spot::tgba\\_reachable\\_iterator](#).

**11.62.4.7** `void spot::parity_game_graph::print (std::ostream & os)`

**11.62.4.8** `virtual void spot::tgba_reachable_iterator::process_link (const state * in_s, int in, const state * out_s, int out, const tgba\_succ\_iterator * si)` [virtual, inherited]

Called by [run\(\)](#) to process a transition.

#### Parameters:

*in\_s* The source state

*in* The source state number.

*out\_s* The destination state

*out* The destination state number.

*si* The [spot::tgba\\_succ\\_iterator](#) positionned on the current transition.

The *in\_s* and *out\_s* states are owned by the [spot::tgba\\_reachable\\_iterator](#) instance and destroyed when the instance is destroyed.

**11.62.4.9** void `spot::parity_game_graph::process_link` (int *in*, int *out*, const [tgba\\_succ\\_iterator](#) \* *si*) [protected]

**11.62.4.10** void `spot::parity_game_graph::process_state` (const [state](#) \* *s*, int *n*, [tgba\\_succ\\_iterator](#) \* *si*) [protected, virtual]

Called by `run()` to process a state.

**Parameters:**

- s* The current state.
- n* A unique number assigned to *s*.
- si* The [spot::tgba\\_succ\\_iterator](#) for *s*.

Reimplemented from [spot::tgba\\_reachable\\_iterator](#).

**11.62.4.11** void `spot::tgba_reachable_iterator::run` () [inherited]

Iterate over all reachable states of a [spot::tgba](#).

This is a template method that will call `add_state()`, `next_state()`, `start()`, `end()`, `process_state()`, and `process_link()`, while it iterate over state.

**11.62.4.12** void `spot::parity_game_graph::start` () [protected, virtual]

Called by `run()` before starting its iteration.

Reimplemented from [spot::tgba\\_reachable\\_iterator](#).

## 11.62.5 Member Data Documentation

**11.62.5.1** const [tgba](#)\* `spot::tgba_reachable_iterator::automata_` [protected, inherited]

The [spot::tgba](#) to explore.

**11.62.5.2** [dn\\_v](#) `spot::parity_game_graph::duplicator_vertice_` [protected]

**11.62.5.3** int `spot::parity_game_graph::nb_node_parity_game` [protected]

**11.62.5.4** [seen\\_map](#) `spot::tgba_reachable_iterator::seen` [protected, inherited]

States already seen.

**11.62.5.5** [sn\\_v](#) `spot::parity_game_graph::spoiler_vertice_` [protected]

**11.62.5.6** [s\\_v](#) `spot::parity_game_graph::tgba_state_` [protected]

### 11.62.5.7 std::deque<const state\*> spot::tgba\_reachable\_iterator\_breadth\_first::todo

[protected, inherited]

A queue of states yet to explore.

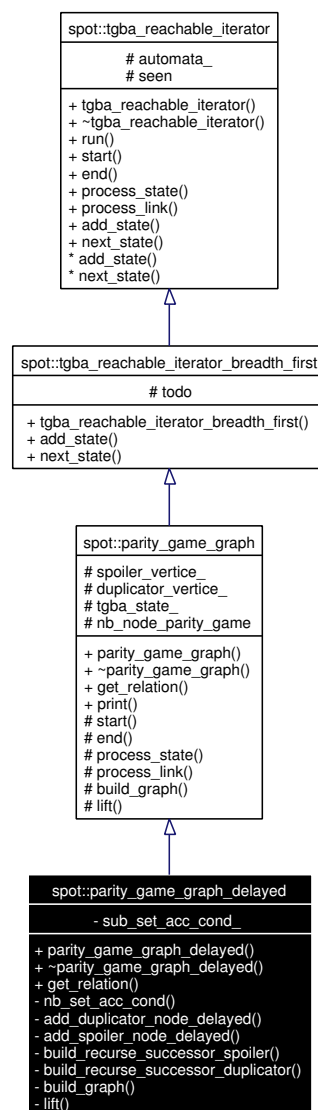
The documentation for this class was generated from the following file:

- tgbaalgos/reductgba\_sim.hh

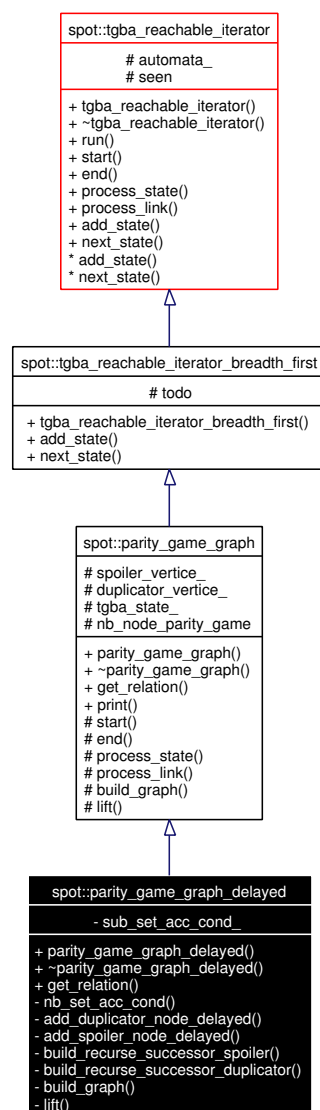
## 11.63 spot::parity\_game\_graph\_delayed Class Reference

```
#include <tgbaalgos/reductgba_sim.hh>
```

Inheritance diagram for spot::parity\_game\_graph\_delayed:



Collaboration diagram for spot::parity\_game\_graph\_delayed:



## Public Member Functions

- [parity\\_game\\_graph\\_delayed](#) (const [tgba](#) \*a)
- [~parity\\_game\\_graph\\_delayed](#) ()
- virtual [delayed\\_simulation\\_relation](#) \* [get\\_relation](#) ()
- void [print](#) (std::ostream &os)
- virtual void [process\\_link](#) (const [state](#) \*in\_s, int in, const [state](#) \*out\_s, int out, const [tgba\\_succ\\_iterator](#) \*si)
- virtual void [add\\_state](#) (const [state](#) \*s)
- virtual const [state](#) \* [next\\_state](#) ()

*Called by [run\(\)](#) to obtain the.*

- void [run](#) ()
- Iterate over all reachable states of a [spot::tgba](#).*

### Protected Types

- typedef Sgi::hash\_map< const [state](#) \*, int, [state\\_ptr\\_hash](#), [state\\_ptr\\_equal](#) > [seen\\_map](#)

### Protected Member Functions

- void [start](#) ()  
*Called by [run\(\)](#) before starting its iteration.*
- void [end](#) ()  
*Called by [run\(\)](#) once all states have been explored.*
- void [process\\_state](#) (const [state](#) \*s, int n, [tgba\\_succ\\_iterator](#) \*si)
- void [process\\_link](#) (int in, int out, const [tgba\\_succ\\_iterator](#) \*si)

### Protected Attributes

- [sn\\_v](#) [spoiler\\_vertice\\_](#)
- [dn\\_v](#) [duplicator\\_vertice\\_](#)
- [s\\_v](#) [tgba\\_state\\_](#)
- int [nb\\_node\\_parity\\_game](#)
- std::deque< const [state](#) \* > [todo](#)  
*A queue of states yet to explore.*
- const [tgba](#) \* [automata\\_](#)  
*The [spot::tgba](#) to explore.*
- [seen\\_map](#) [seen](#)  
*States already seen.*

### Private Types

- typedef Sgi::vector< [bdd](#) > [bdd\\_v](#)

### Private Member Functions

- int [nb\\_set\\_acc\\_cond](#) ()  
*Return the number of acceptance condition.*
- [duplicator\\_node\\_delayed](#) \* [add\\_duplicator\\_node\\_delayed](#) (const [spot::state](#) \*sn, const [spot::state](#) \*dn, [bdd](#) acc, [bdd](#) label, int nb)
- [spoiler\\_node\\_delayed](#) \* [add\\_spoiler\\_node\\_delayed](#) (const [spot::state](#) \*sn, const [spot::state](#) \*dn, [bdd](#) acc, int nb)
- void [build\\_recurse\\_successor\\_spoiler](#) ([spoiler\\_node](#) \*sn, std::ostream &os)
- void [build\\_recurse\\_successor\\_duplicator](#) ([duplicator\\_node](#) \*dn, [spoiler\\_node](#) \*sn, std::ostream &os)
- virtual void [build\\_graph](#) ()  
*Compute the couple as for direct simulation.,.*

- virtual void [lift](#) ()  
*The Jurdzinski's lifting algorithm.*

## Private Attributes

- [bdd\\_v sub\\_set\\_acc\\_cond\\_](#)

### 11.63.1 Detailed Description

Parity game graph which computes the delayed simulation relation as explained in

```
@InProceedings{etessami.01.alp,
  author = {Kousha Etessami and Thomas Wilke and Rebecca A. Schuller},
  title = {Fair Simulation Relations, Parity Games, and State Space
    Reduction for Buchi Automata},
  booktitle = {Proceedings of the 28th international colloquium on
    Automata, Languages and Programming},
  pages = {694--707},
  year = {2001},
  editor = {Fernando Orejas and Paul G. Spirakis and Jan van Leeuwen},
  volume = {2076},
  series = {Lecture Notes in Computer Science},
  address = {Crete, Greece},
  month = {July},
  publisher = {Springer-Verlag}
}
```

### 11.63.2 Member Typedef Documentation

#### 11.63.2.1 typedef Sgi::vector<bdd> [spot::parity\\_game\\_graph\\_delayed::bdd\\_v](#) [private]

Vector which contain all the sub-set of the set of acceptance condition.

#### 11.63.2.2 typedef Sgi::hash\_map<const [state\\*](#), int, [state\\_ptr\\_hash](#), [state\\_ptr\\_equal](#)> [spot::tgba\\_reachable\\_iterator::seen\\_map](#) [protected, inherited]

Reimplemented in [spot::tgba\\_reduc](#).

### 11.63.3 Constructor & Destructor Documentation

#### 11.63.3.1 [spot::parity\\_game\\_graph\\_delayed::parity\\_game\\_graph\\_delayed](#) (const [tgba](#) \* *a*)

#### 11.63.3.2 [spot::parity\\_game\\_graph\\_delayed::~~parity\\_game\\_graph\\_delayed](#) ()

### 11.63.4 Member Function Documentation

#### 11.63.4.1 [duplicator\\_node\\_delayed\\*](#) [spot::parity\\_game\\_graph\\_delayed::add\\_duplicator\\_node\\_delayed](#) (const [spot::state](#) \* *sn*, const [spot::state](#) \* *dn*, bdd *acc*, bdd *label*, int *nb*) [private]

**11.63.4.2** `spoiler_node_delayed*` `spot::parity_game_graph_delayed::add_spoiler_node_delayed` (const `spot::state` \* *sn*, const `spot::state` \* *dn*, bdd *acc*, int *nb*) [private]

**11.63.4.3** `virtual void` `spot::tgba_reachable_iterator_breadth_first::add_state` (const `state` \* *s*) [virtual, inherited]

Implements `spot::tgba_reachable_iterator`.

**11.63.4.4** `virtual void` `spot::parity_game_graph_delayed::build_graph` () [private, virtual]

Compute the couple as for direct simulation,.

Implements `spot::parity_game_graph`.

**11.63.4.5** `void` `spot::parity_game_graph_delayed::build_recurse_successor_duplicator` (`duplicator_node` \* *dn*, `spoiler_node` \* *sn*, `std::ostream` & *os*) [private]

**11.63.4.6** `void` `spot::parity_game_graph_delayed::build_recurse_successor_spoiler` (`spoiler_node` \* *sn*, `std::ostream` & *os*) [private]

**11.63.4.7** `void` `spot::parity_game_graph::end` () [protected, virtual, inherited]

Called by `run()` once all states have been explored.

Reimplemented from `spot::tgba_reachable_iterator`.

**11.63.4.8** `virtual delayed_simulation_relation*` `spot::parity_game_graph_delayed::get_relation` () [virtual]

Implements `spot::parity_game_graph`.

**11.63.4.9** `virtual void` `spot::parity_game_graph_delayed::lift` () [private, virtual]

The Jurdzinski's lifting algorithm.

Implements `spot::parity_game_graph`.

**11.63.4.10** `int` `spot::parity_game_graph_delayed::nb_set_acc_cond` () [private]

Return the number of acceptance condition.

**11.63.4.11** `virtual const state*` `spot::tgba_reachable_iterator_breadth_first::next_state` () [virtual, inherited]

Called by `run()` to obtain the.

Implements `spot::tgba_reachable_iterator`.

**11.63.4.12** `void` `spot::parity_game_graph::print` (`std::ostream` & *os*) [inherited]



**11.63.4.13** `virtual void spot::tgba_reachable_iterator::process_link (const state * in_s, int in, const state * out_s, int out, const tgba\_succ\_iterator * si)` [virtual, inherited]

Called by [run\(\)](#) to process a transition.

**Parameters:**

*in\_s* The source state

*in* The source state number.

*out\_s* The destination state

*out* The destination state number.

*si* The [spot::tgba\\_succ\\_iterator](#) positionned on the current transition.

The *in\_s* and *out\_s* states are owned by the [spot::tgba\\_reachable\\_iterator](#) instance and destroyed when the instance is destroyed.

**11.63.4.14** `void spot::parity_game_graph::process_link (int in, int out, const tgba\_succ\_iterator * si)` [protected, inherited]

**11.63.4.15** `void spot::parity_game_graph::process_state (const state * s, int n, tgba\_succ\_iterator * si)` [protected, virtual, inherited]

Called by [run\(\)](#) to process a state.

**Parameters:**

*s* The current state.

*n* A unique number assigned to *s*.

*si* The [spot::tgba\\_succ\\_iterator](#) for *s*.

Reimplemented from [spot::tgba\\_reachable\\_iterator](#).

**11.63.4.16** `void spot::tgba_reachable_iterator::run ()` [inherited]

Iterate over all reachable states of a [spot::tgba](#).

This is a template method that will call [add\\_state\(\)](#), [next\\_state\(\)](#), [start\(\)](#), [end\(\)](#), [process\\_state\(\)](#), and [process\\_link\(\)](#), while it iterate over state.

**11.63.4.17** `void spot::parity_game_graph::start ()` [protected, virtual, inherited]

Called by [run\(\)](#) before starting its iteration.

Reimplemented from [spot::tgba\\_reachable\\_iterator](#).

## 11.63.5 Member Data Documentation

**11.63.5.1** `const tgba* spot::tgba_reachable_iterator::automata_` [protected, inherited]

The [spot::tgba](#) to explore.

**11.63.5.2** `dn_v spot::parity_game_graph::duplicator_vertice_` [protected, inherited]

**11.63.5.3** `int spot::parity_game_graph::nb_node_parity_game` [protected, inherited]

**11.63.5.4** `seen_map spot::tgba_reachable_iterator::seen` [protected, inherited]

States already seen.

**11.63.5.5** `sn_v spot::parity_game_graph::spoiler_vertice_` [protected, inherited]

**11.63.5.6** `bdd_v spot::parity_game_graph_delayed::sub_set_acc_cond_` [private]

**11.63.5.7** `s_v spot::parity_game_graph::tgba_state_` [protected, inherited]

**11.63.5.8** `std::deque<const state*> spot::tgba_reachable_iterator_breadth_first::todo`  
[protected, inherited]

A queue of states yet to explore.

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

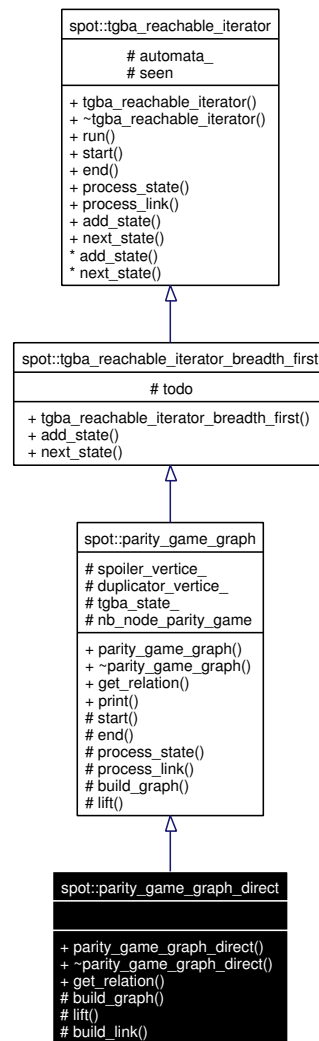
- [tgbaalgos/reductgba\\_sim.hh](#)

## 11.64 spot::parity\_game\_graph\_direct Class Reference

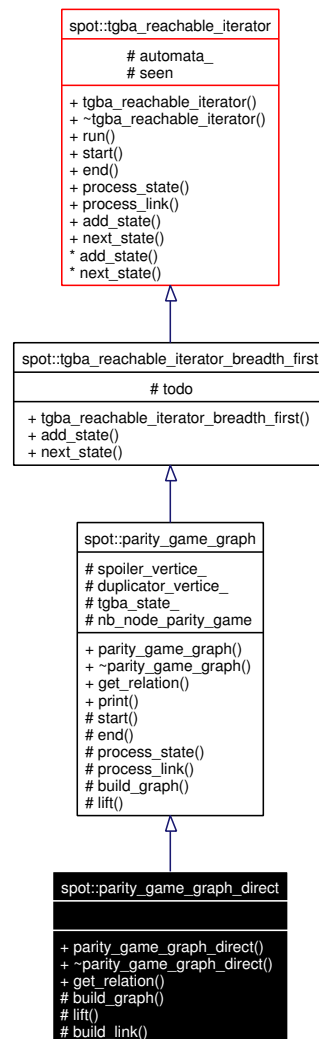
Parity game graph which compute the direct simulation relation.

```
#include <tgbaalgos/reductgba_sim.hh>
```

Inheritance diagram for spot::parity\_game\_graph\_direct:



Collaboration diagram for spot::parity\_game\_graph\_direct:



## Public Member Functions

- [parity\\_game\\_graph\\_direct](#) (const [tgba](#) \*a)
- [~parity\\_game\\_graph\\_direct](#) ()
- virtual [direct\\_simulation\\_relation](#) \* [get\\_relation](#) ()
- void [print](#) (std::ostream &os)
- virtual void [process\\_link](#) (const [state](#) \*in\_s, int in, const [state](#) \*out\_s, int out, const [tgba\\_succ\\_iterator](#) \*si)
- virtual void [add\\_state](#) (const [state](#) \*s)
- virtual const [state](#) \* [next\\_state](#) ()  
*Called by [run\(\)](#) to obtain the.*
- void [run](#) ()  
*Iterate over all reachable states of a [spot::tgba](#).*

### Protected Types

- typedef Sgi::hash\_map< const [state](#) \*, int, [state\\_ptr\\_hash](#), [state\\_ptr\\_equal](#) > [seen\\_map](#)

### Protected Member Functions

- virtual void [build\\_graph](#) ()  
*Compute each node of the graph.*
- virtual void [lift](#) ()  
*Remove edge from spoiler to duplicator that make duplicator loose. Spoiler node whose still have some link, reveal a direct simulation relation.*
- void [build\\_link](#) ()
- void [start](#) ()  
*Called by [run\(\)](#) before starting its iteration.*
- void [end](#) ()  
*Called by [run\(\)](#) once all states have been explored.*
- void [process\\_state](#) (const [state](#) \*s, int n, [tgba\\_succ\\_iterator](#) \*si)
- void [process\\_link](#) (int in, int out, const [tgba\\_succ\\_iterator](#) \*si)

### Protected Attributes

- [sn\\_v](#) [spoiler\\_vertice\\_](#)
- [dn\\_v](#) [duplicator\\_vertice\\_](#)
- [s\\_v](#) [tgba\\_state\\_](#)
- int [nb\\_node\\_parity\\_game](#)
- std::deque< const [state](#) \* > [todo](#)  
*A queue of states yet to explore.*
- const [tgba](#) \* [automata\\_](#)  
*The [spot::tgba](#) to explore.*
- [seen\\_map](#) [seen](#)  
*States already seen.*

#### 11.64.1 Detailed Description

Parity game graph which compute the direct simulation relation.

#### 11.64.2 Member Typedef Documentation

**11.64.2.1** typedef Sgi::hash\_map<const [state](#)\*, int, [state\\_ptr\\_hash](#), [state\\_ptr\\_equal](#)> [spot::tgba\\_reachable\\_iterator::seen\\_map](#) [protected, inherited]

Reimplemented in [spot::tgba\\_reduc](#).

### 11.64.3 Constructor & Destructor Documentation

**11.64.3.1** spot::parity\_game\_graph\_direct::parity\_game\_graph\_direct (const [tgba](#) \* *a*)

**11.64.3.2** spot::parity\_game\_graph\_direct::~~parity\_game\_graph\_direct ()

### 11.64.4 Member Function Documentation

**11.64.4.1** virtual void spot::tgba\_reachable\_iterator\_breadth\_first::add\_state (const [state](#) \* *s*)  
[virtual, inherited]

Implements [spot::tgba\\_reachable\\_iterator](#).

**11.64.4.2** virtual void spot::parity\_game\_graph\_direct::build\_graph () [protected, virtual]

Compute each node of the graph.

Implements [spot::parity\\_game\\_graph](#).

**11.64.4.3** void spot::parity\_game\_graph\_direct::build\_link () [protected]

**11.64.4.4** void spot::parity\_game\_graph::end () [protected, virtual, inherited]

Called by [run\(\)](#) once all states have been explored.

Reimplemented from [spot::tgba\\_reachable\\_iterator](#).

**11.64.4.5** virtual [direct\\_simulation\\_relation](#)\* spot::parity\_game\_graph\_direct::get\_relation ()  
[virtual]

Implements [spot::parity\\_game\\_graph](#).

**11.64.4.6** virtual void spot::parity\_game\_graph\_direct::lift () [protected, virtual]

Remove edge from spoiler to duplicator that make duplicator loose. Spoiler node whose still have some link, reveal a direct simulation relation.

Implements [spot::parity\\_game\\_graph](#).

**11.64.4.7** virtual const [state](#)\* spot::tgba\_reachable\_iterator\_breadth\_first::next\_state ()  
[virtual, inherited]

Called by [run\(\)](#) to obtain the.

Implements [spot::tgba\\_reachable\\_iterator](#).

**11.64.4.8** void spot::parity\_game\_graph::print (std::ostream & *os*) [inherited]

**11.64.4.9** virtual void spot::tgba\_reachable\_iterator::process\_link (const [state](#) \* *in\_s*, int *in*, const [state](#) \* *out\_s*, int *out*, const [tgba\\_succ\\_iterator](#) \* *si*) [virtual, inherited]

Called by [run\(\)](#) to process a transition.

**Parameters:**

- in\_s* The source state
- in* The source state number.
- out\_s* The destination state
- out* The destination state number.
- si* The [spot::tgba\\_succ\\_iterator](#) positionned on the current transition.

The *in\_s* and *out\_s* states are owned by the [spot::tgba\\_reachable\\_iterator](#) instance and destroyed when the instance is destroyed.

**11.64.4.10 void spot::parity\_game\_graph::process\_link (int *in*, int *out*, const [tgba\\_succ\\_iterator](#) \* *si*)** [protected, inherited]

**11.64.4.11 void spot::parity\_game\_graph::process\_state (const [state](#) \* *s*, int *n*, [tgba\\_succ\\_iterator](#) \* *si*)** [protected, virtual, inherited]

Called by [run\(\)](#) to process a state.

**Parameters:**

- s* The current state.
- n* A unique number assigned to *s*.
- si* The [spot::tgba\\_succ\\_iterator](#) for *s*.

Reimplemented from [spot::tgba\\_reachable\\_iterator](#).

**11.64.4.12 void spot::tgba\_reachable\_iterator::run ()** [inherited]

Iterate over all reachable states of a [spot::tgba](#).

This is a template method that will call [add\\_state\(\)](#), [next\\_state\(\)](#), [start\(\)](#), [end\(\)](#), [process\\_state\(\)](#), and [process\\_link\(\)](#), while it iterate over state.

**11.64.4.13 void spot::parity\_game\_graph::start ()** [protected, virtual, inherited]

Called by [run\(\)](#) before starting its iteration.

Reimplemented from [spot::tgba\\_reachable\\_iterator](#).

**11.64.5 Member Data Documentation**

**11.64.5.1 const [tgba](#)\* spot::tgba\_reachable\_iterator::automata\_** [protected, inherited]

The [spot::tgba](#) to explore.

**11.64.5.2 [dn\\_v](#) spot::parity\_game\_graph::duplicator\_vertice\_** [protected, inherited]

**11.64.5.3 int spot::parity\_game\_graph::nb\_node\_parity\_game** [protected, inherited]

**11.64.5.4** [seen\\_map spot::tgba\\_reachable\\_iterator::seen](#) [protected, inherited]

States already seen.

**11.64.5.5** [sn\\_v spot::parity\\_game\\_graph::spoiler\\_vertice\\_](#) [protected, inherited]

**11.64.5.6** [s\\_v spot::parity\\_game\\_graph::tgba\\_state\\_](#) [protected, inherited]

**11.64.5.7** [std::deque<const state\\*> spot::tgba\\_reachable\\_iterator\\_breadth\\_first::todo](#)  
[protected, inherited]

A queue of states yet to explore.

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

- [tgbaalgorithms/reductgba\\_sim.hh](#)

## 11.65 yy::Position Class Reference

Abstract a [Position](#).

```
#include <ltlparse/position.hh>
```

### Public Member Functions

#### Ctor & dtor.

- [Position](#) ()  
*Construct a [Position](#).*

#### Line and Column related manipulators

- void [lines](#) (int count=1)  
*(line related) Advance to the COUNT next lines.*
- void [columns](#) (int count=1)  
*(column related) Advance to the COUNT next columns.*

### Public Attributes

- std::string [filename](#)  
*File name to which this position refers.*
- unsigned int [line](#)  
*Current line number.*
- unsigned int [column](#)  
*Current column number.*



### Static Public Attributes

- static const unsigned int [initial\\_column](#) = 0  
*Initial column number.*
- static const unsigned int [initial\\_line](#) = 1  
*Initial line number.*

### 11.65.1 Detailed Description

Abstract a [Position](#).

### 11.65.2 Constructor & Destructor Documentation

#### 11.65.2.1 yy::Position::Position () [inline]

Construct a [Position](#).

### 11.65.3 Member Function Documentation

#### 11.65.3.1 void yy::Position::columns (int *count* = 1) [inline]

(column related) Advance to the COUNT next columns.

#### 11.65.3.2 void yy::Position::lines (int *count* = 1) [inline]

(line related) Advance to the COUNT next lines.

### 11.65.4 Member Data Documentation

#### 11.65.4.1 unsigned int yy::Position::column

Current column number.

#### 11.65.4.2 std::string yy::Position::filename

File name to which this position refers.

#### 11.65.4.3 const unsigned int yy::Position::initial\_column = 0 [static]

Initial column number.

#### 11.65.4.4 const unsigned int yy::Position::initial\_line = 1 [static]

Initial line number.

#### 11.65.4.5 unsigned int yy::Position::line

Current line number.

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

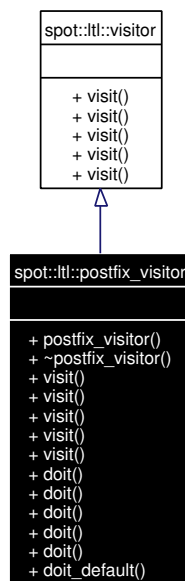
- [ltlparse/position.hh](#)

## 11.66 spot::ltl::postfix\_visitor Class Reference

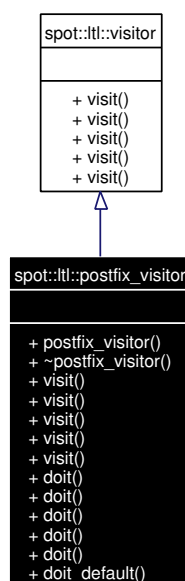
Apply an algorithm on each node of an AST, during a postfix traversal.

```
#include <ltlvisit/postfix.hh>
```

Inheritance diagram for spot::ltl::postfix\_visitor:



Collaboration diagram for spot::ltl::postfix\_visitor:



**Public Member Functions**

- [postfix\\_visitor](#) ()
- virtual [~postfix\\_visitor](#) ()
- void [visit](#) ([atomic\\_prop](#) \*ap)
- void [visit](#) ([unop](#) \*uo)
- void [visit](#) ([binop](#) \*bo)
- void [visit](#) ([multop](#) \*mo)
- void [visit](#) ([constant](#) \*c)
- virtual void [doit](#) ([atomic\\_prop](#) \*ap)
- virtual void [doit](#) ([unop](#) \*uo)
- virtual void [doit](#) ([binop](#) \*bo)
- virtual void [doit](#) ([multop](#) \*mo)
- virtual void [doit](#) ([constant](#) \*c)
- virtual void [doit\\_default](#) ([formula](#) \*f)

**11.66.1 Detailed Description**

Apply an algorithm on each node of an AST, during a postfix traversal.

Override one or more of the postfix\_visitor::doit methods with the algorithm to apply.

**11.66.2 Constructor & Destructor Documentation****11.66.2.1 spot::ltl::postfix\_visitor::postfix\_visitor ()****11.66.2.2 virtual spot::ltl::postfix\_visitor::~~postfix\_visitor () [virtual]****11.66.3 Member Function Documentation****11.66.3.1 virtual void spot::ltl::postfix\_visitor::doit ([constant](#) \*c) [virtual]****11.66.3.2 virtual void spot::ltl::postfix\_visitor::doit ([multop](#) \*mo) [virtual]****11.66.3.3 virtual void spot::ltl::postfix\_visitor::doit ([binop](#) \*bo) [virtual]****11.66.3.4 virtual void spot::ltl::postfix\_visitor::doit ([unop](#) \*uo) [virtual]****11.66.3.5 virtual void spot::ltl::postfix\_visitor::doit ([atomic\\_prop](#) \*ap) [virtual]****11.66.3.6 virtual void spot::ltl::postfix\_visitor::doit\_default ([formula](#) \*f) [virtual]****11.66.3.7 void spot::ltl::postfix\_visitor::visit ([constant](#) \*c) [virtual]**

Implements [spot::ltl::visitor](#).

**11.66.3.8** `void spot::ltl::postfix_visitor::visit (multop * mo) [virtual]`

Implements [spot::ltl::visitor](#).

**11.66.3.9** `void spot::ltl::postfix_visitor::visit (binop * bo) [virtual]`

Implements [spot::ltl::visitor](#).

**11.66.3.10** `void spot::ltl::postfix_visitor::visit (unop * uo) [virtual]`

Implements [spot::ltl::visitor](#).

**11.66.3.11** `void spot::ltl::postfix_visitor::visit (atomic_prop * ap) [virtual]`

Implements [spot::ltl::visitor](#).

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

- [ltlvisit/postfix.hh](#)

## 11.67 `spot::ptr_hash< T >` Struct Template Reference

A hash function for pointers.

```
#include <misc/hash.hh>
```

### Public Member Functions

- `size_t operator() (const T *p) const`

#### 11.67.1 Detailed Description

```
template<class T> struct spot::ptr_hash< T >
```

A hash function for pointers.

#### 11.67.2 Member Function Documentation

**11.67.2.1** `template<class T> size_t spot::ptr_hash< T >::operator() (const T * p) const [inline]`

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

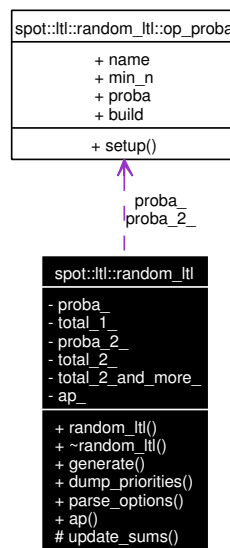
- [misc/hash.hh](#)

## 11.68 `spot::ltl::random_ltl` Class Reference

Generate random LTL formulae.

```
#include <ltlvisit/randomltl.hh>
```

Collaboration diagram for `spot::ltl::random_ltl`:



## Public Member Functions

- `random_ltl (const atomic_prop_set *ap)`  
Create a random LTL generator using atomic propositions from ap.
- `~random_ltl ()`
- `formula * generate (int n) const`  
Generate a formula of size n.
- `std::ostream & dump_priorities (std::ostream &os) const`  
Print the priorities of each operator, constants, and atomic propositions.
- `const char * parse_options (char *options)`  
Update the priorities used to generate LTL formulae.
- `const atomic_prop_set * ap () const`  
Return the set of atomic proposition used to build formulae.

## Protected Member Functions

- `void update_sums ()`

## Private Attributes

- `op_proba * proba_`
- `double total_1_`
- `op_proba * proba_2_`
- `double total_2_`
- `double total_2_and_more_`
- `const atomic_prop_set * ap_`

## Classes

- struct [op\\_proba](#)

## 11.68.1 Detailed Description

Generate random LTL formulae.

This class recursively construct LTL formulae of a given size. The formulae will use the use atomic propositions from the set of proposition passed to the constructor, in addition to the constant and all LTL operators supported by Spot.

By default each operator has equal chance to be selected. Also, each atomic proposition has as much chance as each constant (i.e., true and false) to be picked. This can be tuned using [parse\\_options\(\)](#).

## 11.68.2 Constructor &amp; Destructor Documentation

11.68.2.1 `spot::ltl::random_ltl::random_ltl (const atomic\_prop\_set * ap)`

Create a random LTL generator using atomic propositions from *ap*.

11.68.2.2 `spot::ltl::random_ltl::~~random_ltl ()`

## 11.68.3 Member Function Documentation

11.68.3.1 `const atomic\_prop\_set* spot::ltl::random_ltl::ap () const` `[inline]`

Return the set of atomic proposition used to build formulae.

11.68.3.2 `std::ostream& spot::ltl::random_ltl::dump_priorities (std::ostream & os) const`

Print the priorities of each operator, constants, and atomic propositions.

11.68.3.3 `formula* spot::ltl::random_ltl::generate (int n) const`

Generate a formula of size *n*.

It is possible to obtain formulae that are smaller than *n*, because some simple simplifications are performed by the AST. (For instance the formula `a | a` is automatically reduced to `a` by [spot::ltl::multop](#).)

Furthermore, for the purpose of this generator, `a | b | c` has length 5, while it has length 4 for [spot::ltl::length](#)).

11.68.3.4 `const char* spot::ltl::random_ltl::parse_options (char * options)`

Update the priorities used to generate LTL formulae.

The initial priorities are as follows.

<code>ap</code>	<code>n</code>
<code>false</code>	<code>1</code>
<code>true</code>	<code>1</code>
<code>not</code>	<code>1</code>
<code>F</code>	<code>1</code>
<code>G</code>	<code>1</code>

<code>x</code>	1
<code>equiv</code>	1
<code>implies</code>	1
<code>xor</code>	1
<code>R</code>	1
<code>U</code>	1
<code>and</code>	1
<code>or</code>	1

Where `n` is the number of atomic propositions in the set passed to the constructor.

This means that each operator has equal chance to be selected. Also, each atomic proposition has as much chance as each constant (i.e., true and false) to be picked. This can be

These priorities can be altered using this function. *options* should be comma-separated list of KEY=VALUE assignments, using keys from the above list. For instance "`xor=0, F=3`" will prevent `xor` from being used, and will raise the relative probability of occurrences of the `F` operator.

**11.68.3.5** `void spot::ltl::random_ltl::update_sums ()` [protected]

#### 11.68.4 Member Data Documentation

**11.68.4.1** `const atomic_prop_set* spot::ltl::random_ltl::ap_` [private]

**11.68.4.2** `op_proba* spot::ltl::random_ltl::proba_` [private]

**11.68.4.3** `op_proba* spot::ltl::random_ltl::proba_2_` [private]

**11.68.4.4** `double spot::ltl::random_ltl::total_1_` [private]

**11.68.4.5** `double spot::ltl::random_ltl::total_2_` [private]

**11.68.4.6** `double spot::ltl::random_ltl::total_2_and_more_` [private]

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

- `ltlvisit/randomltl.hh`

## 11.69 `spot::ltl::random_ltl::op_proba` Struct Reference

### Public Types

- typedef `formula` `*(builder)`(const `random_ltl` \*rl, int n)

### Public Member Functions

- void `setup` (const char \*name, int min\_n, `builder` build)

**Public Attributes**

- const char \* [name](#)
- int [min\\_n](#)
- double [proba](#)
- [builder](#) [build](#)

**11.69.1 Member Typedef Documentation**

**11.69.1.1** typedef [formula](#)\*([\\* spot::ltl::random\\_ltl::op\\_proba::builder](#))(const [random\\_ltl](#) \*rl, int n)

**11.69.2 Member Function Documentation**

**11.69.2.1** void [spot::ltl::random\\_ltl::op\\_proba::setup](#) (const char \* *name*, int *min\_n*, [builder](#) *build*)

**11.69.3 Member Data Documentation**

**11.69.3.1** [builder](#) [spot::ltl::random\\_ltl::op\\_proba::build](#)

**11.69.3.2** int [spot::ltl::random\\_ltl::op\\_proba::min\\_n](#)

**11.69.3.3** const char\* [spot::ltl::random\\_ltl::op\\_proba::name](#)

**11.69.3.4** double [spot::ltl::random\\_ltl::op\\_proba::proba](#)

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

- [ltlvisit/randomltl.hh](#)

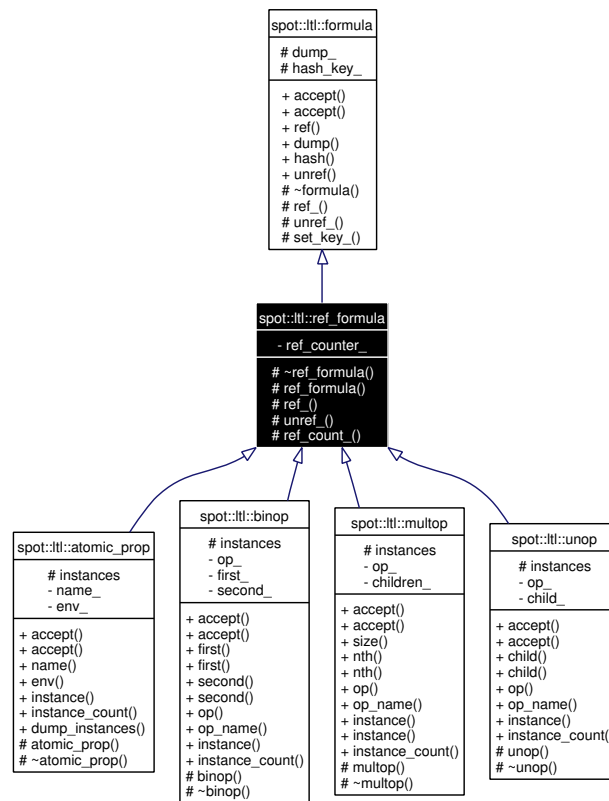
**11.70 spot::ltl::ref\_formula Class Reference**

A reference-counted LTL formula.

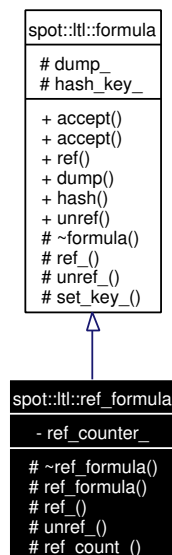
```
#include <ltlast/refformula.hh>
```

Inheritance diagram for `spot::ltl::ref_formula`:





Collaboration diagram for spot::ltl::ref\_formula:



## Public Member Functions

- virtual void `accept` (visitor &v)=0

*Entry point for vspot::ltl::visitor instances.*

- virtual void **accept** (const\_visitor &v) const =0  
*Entry point for vspot::ltl::const\_visitor instances.*
- **formula** \* **ref** ()  
*clone this node*
- const std::string & **dump** () const  
*Return a canonic representation of the formula.*
- const size\_t **hash** () const  
*Return a hash\_key for the formula.*

### Static Public Member Functions

- static void **unref** (formula \*f)  
*release this node*

### Protected Member Functions

- virtual ~**ref\_formula** ()
- **ref\_formula** ()
- void **ref\_** ()  
*increment reference counter if any*
- bool **unref\_** ()  
*decrement reference counter if any, return true when the instance must be deleted (usually when the counter hits 0).*
- unsigned **ref\_count\_** ()  
*Number of references to this formula.*
- void **set\_key\_** ()  
*Compute key\_ from dump\_.*

### Protected Attributes

- std::string **dump\_**  
*The canonic representation of the formula.*
- size\_t **hash\_key\_**  
*The hash key of this formula.*

**Private Attributes**

- unsigned `ref_counter_`

**11.70.1 Detailed Description**

A reference-counted LTL formula.

**11.70.2 Constructor & Destructor Documentation**

**11.70.2.1** `virtual spot::ltl::ref_formula::~~ref_formula ()` [protected, virtual]

**11.70.2.2** `spot::ltl::ref_formula::ref_formula ()` [protected]

**11.70.3 Member Function Documentation**

**11.70.3.1** `virtual void spot::ltl::formula::accept (const_visitor & v) const` [pure virtual, inherited]

Entry point for `vspot::ltl::const_visitor` instances.

Implemented in `spot::ltl::atomic_prop`, `spot::ltl::binop`, `spot::ltl::constant`, `spot::ltl::multop`, and `spot::ltl::unop`.

**11.70.3.2** `virtual void spot::ltl::formula::accept (visitor & v)` [pure virtual, inherited]

Entry point for `vspot::ltl::visitor` instances.

Implemented in `spot::ltl::atomic_prop`, `spot::ltl::binop`, `spot::ltl::constant`, `spot::ltl::multop`, and `spot::ltl::unop`.

**11.70.3.3** `const std::string& spot::ltl::formula::dump () const` [inherited]

Return a canonic representation of the formula.

**11.70.3.4** `const size_t spot::ltl::formula::hash () const` [inline, inherited]

Return a `hash_key` for the formula.

**11.70.3.5** `formula* spot::ltl::formula::ref ()` [inherited]

clone this node

This increments the reference counter of this node (if one is used). You should almost never use this method directly as it doesn't touch the children. If you want to clone a whole formula, use `spot::ltl::clone()` instead.

**11.70.3.6** `void spot::ltl::ref_formula::ref_ ()` [protected, virtual]

increment reference counter if any

Reimplemented from `spot::ltl::formula`.

**11.70.3.7 unsigned spot::ltl::ref\_formula::ref\_count\_ ()** [protected]

Number of references to this formula.

**11.70.3.8 void spot::ltl::formula::set\_key\_ ()** [protected, inherited]

Compute key\_ from dump\_.

Should be called once in each object, after dump\_ has been set.

**11.70.3.9 static void spot::ltl::formula::unref (formula \*f)** [static, inherited]

release this node

This decrements the reference counter of this node (if one is used) and can free the object. You should almost never use this method directly as it doesn't touch the children. If you want to release a whole formula, use [spot::ltl::destroy\(\)](#) instead.

**11.70.3.10 bool spot::ltl::ref\_formula::unref\_ ()** [protected, virtual]

decrement reference counter if any, return true when the instance must be deleted (usually when the counter hits 0).

Reimplemented from [spot::ltl::formula](#).

**11.70.4 Member Data Documentation****11.70.4.1 std::string spot::ltl::formula::dump\_** [protected, inherited]

The canonic representation of the formula.

**11.70.4.2 size\_t spot::ltl::formula::hash\_key\_** [protected, inherited]

The hash key of this formula.

Initialized by [set\\_key\\_\(\)](#).

**11.70.4.3 unsigned spot::ltl::ref\_formula::ref\_counter\_** [private]

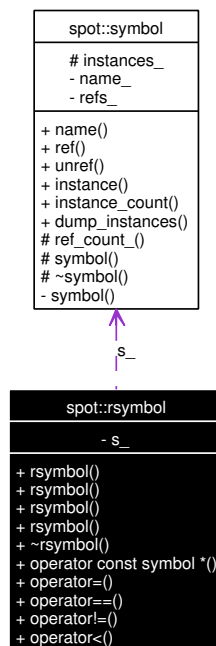
The documentation for this class was generated from the following file:

- [ltlast/refformula.hh](#)

**11.71 spot::rsymbol Class Reference**

```
#include <evtgba/symbol.hh>
```

Collaboration diagram for spot::rsymbol:



### Public Member Functions

- [rsymbol](#) (const [symbol](#) \*s)
- [rsymbol](#) (const std::string &s)
- [rsymbol](#) (const char \*s)
- [rsymbol](#) (const [rsymbol](#) &rs)
- [~rsymbol](#) ()
- [operator const symbol \\* \(\)](#) const
- const [rsymbol](#) & [operator=](#) (const [rsymbol](#) &rs)
- bool [operator==](#) (const [rsymbol](#) &rs) const
- bool [operator!=](#) (const [rsymbol](#) &rs) const
- bool [operator<](#) (const [rsymbol](#) &rs) const

### Private Attributes

- const [symbol](#) \* [s\\_](#)

### 11.71.1 Constructor & Destructor Documentation

**11.71.1.1** `spot::rsymbol::rsymbol (const symbol *s) [inline]`

**11.71.1.2** `spot::rsymbol::rsymbol (const std::string &s) [inline]`

**11.71.1.3** `spot::rsymbol::rsymbol (const char *s) [inline]`

**11.71.1.4** `spot::rsymbol::rsymbol (const rsymbol &rs) [inline]`

11.71.1.5 spot::rsymbol::~~rsymbol () [inline]

### 11.71.2 Member Function Documentation

11.71.2.1 spot::rsymbol::operator const rsymbol \* () const [inline]

11.71.2.2 bool spot::rsymbol::operator!= (const rsymbol & rs) const [inline]

11.71.2.3 bool spot::rsymbol::operator< (const rsymbol & rs) const [inline]

11.71.2.4 const rsymbol& spot::rsymbol::operator= (const rsymbol & rs) [inline]

11.71.2.5 bool spot::rsymbol::operator== (const rsymbol & rs) const [inline]

### 11.71.3 Member Data Documentation

11.71.3.1 const rsymbol\* spot::rsymbol::s\_ [private]

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

- evtgba/symbol.hh

## 11.72 spot::scc\_stack Class Reference

```
#include <tgbaalgorithms/gtec/sccstack.hh>
```

### Public Types

- typedef std::list< connected\_component > stack\_type

### Public Member Functions

- void push (int index)  
*Stack a new SCC with index index.*
- connected\_component & top ()  
*Access the top SCC.*
- const connected\_component & top () const  
*Access the top SCC.*
- void pop ()  
*Pop the top SCC.*
- size\_t size () const  
*How many SCC are in stack.*

- `std::list< const state * > & rem ()`

*The `rem` member of the top SCC.*

- `unsigned clear\_rem \(\)`
- `bool empty \(\) const`

*Is the stack empty?*

### Public Attributes

- `stack\_type s`

### Classes

- `struct connected\_component`

#### 11.72.1 Member Typedef Documentation

**11.72.1.1** `typedef std::list<connected\_component> spot::scc\_stack::stack\_type`

#### 11.72.2 Member Function Documentation

**11.72.2.1** `unsigned spot::scc\_stack::clear\_rem \(\)`

Purge all `rem` members.

#### Returns:

the number of elements cleared.

**11.72.2.2** `bool spot::scc\_stack::empty \(\) const`

Is the stack empty?

**11.72.2.3** `void spot::scc\_stack::pop \(\)`

Pop the top SCC.

**11.72.2.4** `void spot::scc\_stack::push \(int index\)`

Stack a new SCC with index *index*.

**11.72.2.5** `std::list<const state*>& spot::scc\_stack::rem \(\)`

The `rem` member of the top SCC.

**11.72.2.6** `size_t spot::scc\_stack::size \(\) const`

How many SCC are in stack.

**11.72.2.7** const [connected\\_component](#)& spot::scc\_stack::top () const

Access the top SCC.

**11.72.2.8** [connected\\_component](#)& spot::scc\_stack::top ()

Access the top SCC.

**11.72.3** Member Data Documentation**11.72.3.1** [stack\\_type](#) spot::scc\_stack::s

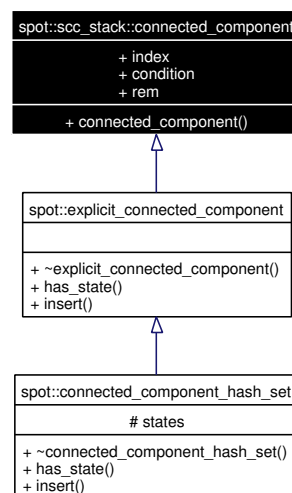
The documentation for this class was generated from the following file:

- [tgbaalgos/gtec/sccstack.hh](#)

**11.73** spot::scc\_stack::connected\_component Struct Reference

```
#include <tgbaalgos/gtec/sccstack.hh>
```

Inheritance diagram for spot::scc\_stack::connected\_component:

**Public Member Functions**

- [connected\\_component](#) (int [index](#)=-1)

**Public Attributes**

- int [index](#)  
*Index of the SCC.*
- bdd [condition](#)
- std::list< const [state](#) \* > [rem](#)



### 11.73.1 Constructor & Destructor Documentation

**11.73.1.1** `spot::scc_stack::connected_component::connected_component (int index = -1)`

### 11.73.2 Member Data Documentation

**11.73.2.1** `bdd spot::scc_stack::connected_component::condition`

The bdd condition is the union of all acceptance conditions of transitions which connect the states of the connected component.

**11.73.2.2** `int spot::scc_stack::connected_component::index`

Index of the SCC.

**11.73.2.3** `std::list<const state*> spot::scc_stack::connected_component::rem`

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

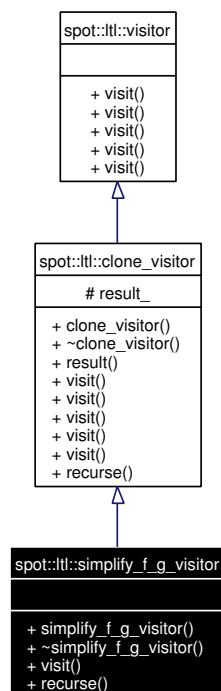
- [tgbaalgos/gtec/sccstack.hh](#)

## 11.74 spot::ltl::simplify\_f\_g\_visitor Class Reference

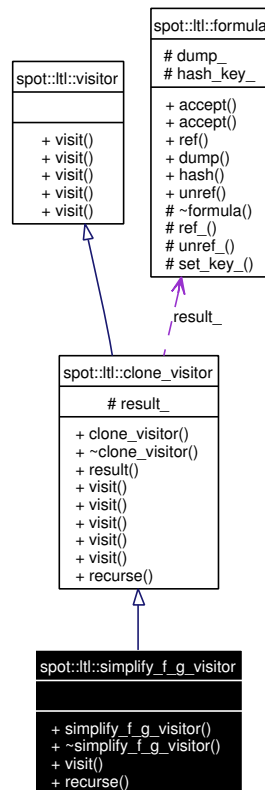
Replace true U f and false R g by F f and G g.

```
#include <ltlvisit/simpfg.hh>
```

Inheritance diagram for spot::ltl::simplify\_f\_g\_visitor:



Collaboration diagram for spot::ltl::simplify\_f\_g\_visitor:



## Public Member Functions

- [simplify\\_f\\_g\\_visitor\(\)](#)
- [virtual ~simplify\\_f\\_g\\_visitor\(\)](#)
- [void visit\(binop \\*bo\)](#)
- [virtual formula \\* recurse\(formula \\*f\)](#)
- [formula \\* result\(\)](#) const
- [void visit\(atomic\\_prop \\*ap\)](#)
- [void visit\(unop \\*uo\)](#)
- [void visit\(multop \\*mo\)](#)
- [void visit\(constant \\*c\)](#)

## Protected Attributes

- [formula \\* result\\_](#)

## Private Types

- [typedef clone\\_visitor super](#)

### 11.74.1 Detailed Description

Replace `true`  $\cup$  `f` and `false`  $\cap$  `g` by `F` `f` and `G` `g`.

### 11.74.2 Member Typedef Documentation

11.74.2.1 `typedef` `clone_visitor` `spot::ltl::simplify_f_g_visitor::super` [private]

### 11.74.3 Constructor & Destructor Documentation

11.74.3.1 `spot::ltl::simplify_f_g_visitor::simplify_f_g_visitor()`

11.74.3.2 `virtual` `spot::ltl::simplify_f_g_visitor::~~simplify_f_g_visitor()` [virtual]

### 11.74.4 Member Function Documentation

11.74.4.1 `virtual` `formula*` `spot::ltl::simplify_f_g_visitor::recurse(formula *f)` [virtual]

Reimplemented from `spot::ltl::clone_visitor`.

11.74.4.2 `formula*` `spot::ltl::clone_visitor::result()` `const` [inherited]

11.74.4.3 `void` `spot::ltl::clone_visitor::visit(constant *c)` [virtual, inherited]

Implements `spot::ltl::visitor`.

11.74.4.4 `void` `spot::ltl::clone_visitor::visit(multop *mo)` [virtual, inherited]

Implements `spot::ltl::visitor`.

11.74.4.5 `void` `spot::ltl::clone_visitor::visit(unop *uo)` [virtual, inherited]

Implements `spot::ltl::visitor`.

Reimplemented in `spot::ltl::unabbreviate_ltl_visitor`.

11.74.4.6 `void` `spot::ltl::clone_visitor::visit(atomic_prop *ap)` [virtual, inherited]

Implements `spot::ltl::visitor`.

11.74.4.7 `void` `spot::ltl::simplify_f_g_visitor::visit(binop *bo)` [virtual]

Reimplemented from `spot::ltl::clone_visitor`.

### 11.74.5 Member Data Documentation

11.74.5.1 `formula*` `spot::ltl::clone_visitor::result_` [protected, inherited]

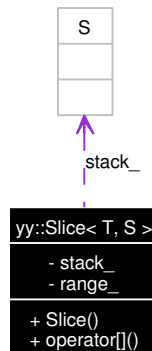
The documentation for this class was generated from the following file:

- `ltlvisit/simpfg.hh`

## 11.75 yy::Slice< T, S > Class Template Reference

```
#include <ltlparse/stack.hh>
```

Collaboration diagram for yy::Slice< T, S >:



### Public Member Functions

- [Slice](#) (const S &stack, unsigned int range)
- const T & [operator\[\]](#) (unsigned int i) const

### Private Attributes

- const S & [stack\\_](#)
- unsigned int [range\\_](#)

```
template<class T, class S = Stack< T >> class yy::Slice< T, S >
```

### 11.75.1 Constructor & Destructor Documentation

**11.75.1.1** template<class T, class S = Stack< T >> [yy::Slice< T, S >::Slice](#) (const S & *stack*, unsigned int *range*) [inline]

### 11.75.2 Member Function Documentation

**11.75.2.1** [ ] template<class T, class S = Stack< T >> const T& [yy::Slice< T, S >::operator\[\]](#) (unsigned int *i*) const [inline]

### 11.75.3 Member Data Documentation

**11.75.3.1** template<class T, class S = Stack< T >> unsigned int [yy::Slice< T, S >::range\\_](#) [private]

**11.75.3.2** template<class T, class S = Stack< T >> const S& [yy::Slice< T, S >::stack\\_](#) [private]

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

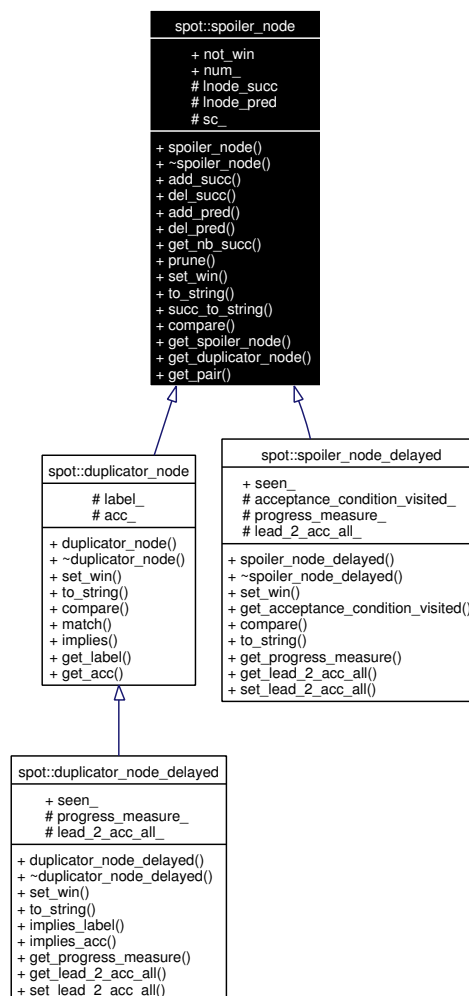
- [ltparse/stack.hh](#)

## 11.76 spot::spoiler\_node Class Reference

Spoiler node of parity game graph.

```
#include <tgbaalgos/reductgba_sim.hh>
```

Inheritance diagram for spot::spoiler\_node:



### Public Member Functions

- `spoiler_node` (const `state` \*d\_node, const `state` \*s\_node, int num)
- virtual `~spoiler_node` ()
- bool `add_succ` (`spoiler_node` \*n)  
Add a successor. Return true if n wasn't yet in the list of successor, false otherwise.
- void `del_succ` (`spoiler_node` \*n)
- virtual void `add_pred` (`spoiler_node` \*n)

- virtual void [del\\_pred](#) ()
- int [get\\_nb\\_succ](#) ()
- bool [prune](#) ()
- virtual bool [set\\_win](#) ()
- virtual std::string [to\\_string](#) (const [tgba](#) \*a)
- virtual std::string [succ\\_to\\_string](#) ()
- virtual bool [compare](#) ([spoiler\\_node](#) \*n)
- const [state](#) \* [get\\_spoiler\\_node](#) ()
- const [state](#) \* [get\\_duplicator\\_node](#) ()
- [state\\_couple](#) \* [get\\_pair](#) ()

### Public Attributes

- bool [not\\_win](#)
- int [num\\_](#)

### Protected Attributes

- [sn\\_v](#) \* [lnode\\_succ](#)
- [sn\\_v](#) \* [lnode\\_pred](#)
- [state\\_couple](#) \* [sc\\_](#)

#### 11.76.1 Detailed Description

Spoiler node of parity game graph.

#### 11.76.2 Constructor & Destructor Documentation

**11.76.2.1** `spot::spoiler_node::spoiler_node (const state * d_node, const state * s_node, int num)`

**11.76.2.2** `virtual spot::spoiler_node::~spoiler\_node ()` [virtual]

#### 11.76.3 Member Function Documentation

**11.76.3.1** `virtual void spot::spoiler_node::add_pred (spoiler\_node * n)` [virtual]

**11.76.3.2** `bool spot::spoiler_node::add_succ (spoiler\_node * n)`

Add a successor. Return true if *n* wasn't yet in the list of successor, false otherwise.

**11.76.3.3** `virtual bool spot::spoiler_node::compare (spoiler\_node * n)` [virtual]

Reimplemented in [spot::duplicator\\_node](#), and [spot::spoiler\\_node\\_delayed](#).

**11.76.3.4** `virtual void spot::spoiler_node::del_pred ()` [virtual]

**11.76.3.5** `void spot::spoiler_node::del_succ (spoiler\_node * n)`

11.76.3.6 `const state* spot::spoiler_node::get_duplicator_node ()`

11.76.3.7 `int spot::spoiler_node::get_nb_succ ()`

11.76.3.8 `state_couple* spot::spoiler_node::get_pair ()`

11.76.3.9 `const state* spot::spoiler_node::get_spoiler_node ()`

11.76.3.10 `bool spot::spoiler_node::prune ()`

11.76.3.11 `virtual bool spot::spoiler_node::set_win ()` [virtual]

Reimplemented in `spot::duplicator_node`, `spot::spoiler_node_delayed`, and `spot::duplicator_node_delayed`.

11.76.3.12 `virtual std::string spot::spoiler_node::succ_to_string ()` [virtual]

11.76.3.13 `virtual std::string spot::spoiler_node::to_string (const tgba *a)` [virtual]

Reimplemented in `spot::duplicator_node`, `spot::spoiler_node_delayed`, and `spot::duplicator_node_delayed`.

## 11.76.4 Member Data Documentation

11.76.4.1 `sn_v* spot::spoiler_node::lnode_pred` [protected]

11.76.4.2 `sn_v* spot::spoiler_node::lnode_succ` [protected]

11.76.4.3 `bool spot::spoiler_node::not_win`

11.76.4.4 `int spot::spoiler_node::num_`

11.76.4.5 `state_couple* spot::spoiler_node::sc_` [protected]

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

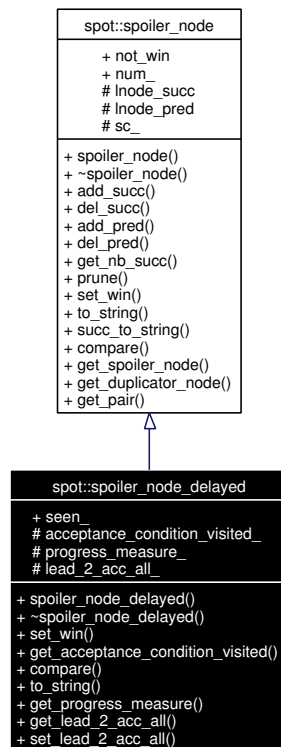
- `tgbaalgos/reductgba_sim.hh`

## 11.77 `spot::spoiler_node_delayed` Class Reference

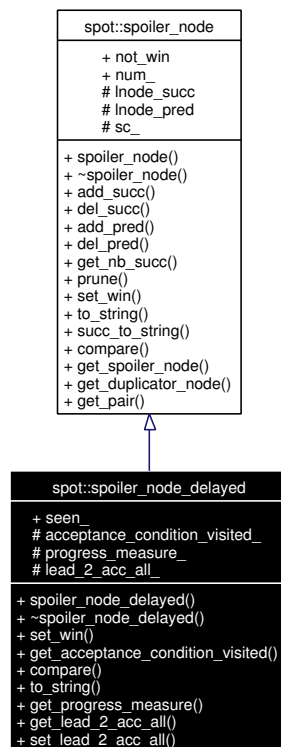
Spoiler node of parity game graph for delayed simulation.

```
#include <tgbaalgos/reductgba_sim.hh>
```

Inheritance diagram for `spot::spoiler_node_delayed`:



Collaboration diagram for `spot::spoiler_node_delayed`:





### Public Member Functions

- `spoiler_node_delayed` (const `state` \*`d_node`, const `state` \*`s_node`, bdd `a`, int `num`)
- `~spoiler_node_delayed` ()
- bool `set_win` ()

*Return true if the progress\_measure has changed.*

- bdd `get_acceptance_condition_visited` () const
- virtual bool `compare` (`spoiler_node` \*`n`)
- virtual std::string `to_string` (const `tgba` \*`a`)
- int `get_progress_measure` () const
- bool `get_lead_2_acc_all` ()
- bool `set_lead_2_acc_all` (bdd `acc`=bddfalse)
- bool `add_succ` (`spoiler_node` \*`n`)

*Add a successor. Return true if n wasn't yet in the list of successor, false otherwise.*

- void `del_succ` (`spoiler_node` \*`n`)
- virtual void `add_pred` (`spoiler_node` \*`n`)
- virtual void `del_pred` ()
- int `get_nb_succ` ()
- bool `prune` ()
- virtual std::string `succ_to_string` ()
- const `state` \* `get_spoiler_node` ()
- const `state` \* `get_duplicator_node` ()
- `state_couple` \* `get_pair` ()

### Public Attributes

- bool `seen_`
- bool `not_win`
- int `num_`

### Protected Attributes

- bdd `acceptance_condition_visited_`
- int `progress_measure_`
- bool `lead_2_acc_all_`
- `sn_v` \* `lnode_succ`
- `sn_v` \* `lnode_pred`
- `state_couple` \* `sc_`

#### 11.77.1 Detailed Description

Spoiler node of parity game graph for delayed simulation.

#### 11.77.2 Constructor & Destructor Documentation

**11.77.2.1** `spot::spoiler_node_delayed::spoiler_node_delayed` (const `state` \* `d_node`, const `state` \* `s_node`, bdd `a`, int `num`)

11.77.2.2 `spot::spoiler_node_delayed::~spoiler_node_delayed ()`

### 11.77.3 Member Function Documentation

11.77.3.1 `virtual void spot::spoiler_node::add_pred (spoiler_node * n) [virtual, inherited]`

11.77.3.2 `bool spot::spoiler_node::add_succ (spoiler_node * n) [inherited]`

Add a successor. Return true if *n* wasn't yet in the list of successor, false otherwise.

11.77.3.3 `virtual bool spot::spoiler_node_delayed::compare (spoiler_node * n) [virtual]`

Reimplemented from `spot::spoiler_node`.

11.77.3.4 `virtual void spot::spoiler_node::del_pred () [virtual, inherited]`

11.77.3.5 `void spot::spoiler_node::del_succ (spoiler_node * n) [inherited]`

11.77.3.6 `bdd spot::spoiler_node_delayed::get_acceptance_condition_visited () const`

11.77.3.7 `const state* spot::spoiler_node::get_duplicator_node () [inherited]`

11.77.3.8 `bool spot::spoiler_node_delayed::get_lead_2_acc_all ()`

11.77.3.9 `int spot::spoiler_node::get_nb_succ () [inherited]`

11.77.3.10 `state_couple* spot::spoiler_node::get_pair () [inherited]`

11.77.3.11 `int spot::spoiler_node_delayed::get_progress_measure () const`

11.77.3.12 `const state* spot::spoiler_node::get_spoiler_node () [inherited]`

11.77.3.13 `bool spot::spoiler_node::prune () [inherited]`

11.77.3.14 `bool spot::spoiler_node_delayed::set_lead_2_acc_all (bdd acc = bddfalse)`

11.77.3.15 `bool spot::spoiler_node_delayed::set_win () [virtual]`

Return true if the `progress_measure` has changed.

Reimplemented from `spot::spoiler_node`.

11.77.3.16 `virtual std::string spot::spoiler_node::succ_to_string () [virtual, inherited]`

**11.77.3.17** virtual std::string spot::spoiler\_node\_delayed::to\_string (const [tgba](#) \* a) [virtual]

Reimplemented from [spot::spoiler\\_node](#).

#### 11.77.4 Member Data Documentation

**11.77.4.1** bdd [spot::spoiler\\_node\\_delayed::acceptance\\_condition\\_visited\\_](#) [protected]

a Bdd for retain all the acceptance condition that a node has visited.

**11.77.4.2** bool [spot::spoiler\\_node\\_delayed::lead\\_2\\_acc\\_all\\_](#) [protected]

**11.77.4.3** [sn\\_v](#)\* [spot::spoiler\\_node::lnode\\_pred](#) [protected, inherited]

**11.77.4.4** [sn\\_v](#)\* [spot::spoiler\\_node::lnode\\_succ](#) [protected, inherited]

**11.77.4.5** bool [spot::spoiler\\_node::not\\_win](#) [inherited]

**11.77.4.6** int [spot::spoiler\\_node::num\\_](#) [inherited]

**11.77.4.7** int [spot::spoiler\\_node\\_delayed::progress\\_measure\\_](#) [protected]

**11.77.4.8** [state\\_couple](#)\* [spot::spoiler\\_node::sc\\_](#) [protected, inherited]

**11.77.4.9** bool [spot::spoiler\\_node\\_delayed::seen\\_](#)

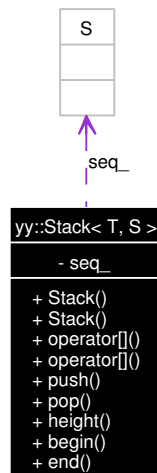
The documentation for this class was generated from the following file:

- [tgbaalgos/reductgba\\_sim.hh](#)

## 11.78 yy::Stack< T, S > Class Template Reference

```
#include <ltlparse/stack.hh>
```

Collaboration diagram for yy::Stack< T, S >:



### Public Types

- typedef S::iterator [Iterator](#)
- typedef S::const\_iterator [ConstIterator](#)

### Public Member Functions

- [Stack](#) ()
- [Stack](#) (unsigned int n)
- T & [operator](#)[] (unsigned int i)
- const T & [operator](#)[] (unsigned int i) const
- void [push](#) (const T &t)
- void [pop](#) (unsigned int n=1)
- unsigned int [height](#) () const
- [ConstIterator](#) [begin](#) () const
- [ConstIterator](#) [end](#) () const

### Private Attributes

- S [seq\\_](#)

```
template<class T, class S = std::deque< T >> class yy::Stack< T, S >
```

#### 11.78.1 Member Typedef Documentation

**11.78.1.1** template<class T, class S = std::deque< T >> typedef S::const\_iterator [yy::Stack](#)< T, S >::[ConstIterator](#)

**11.78.1.2** template<class T, class S = std::deque< T >> typedef S::iterator [yy::Stack](#)< T, S >::[Iterator](#)

## 11.78.2 Constructor & Destructor Documentation

**11.78.2.1** `template<class T, class S = std::deque< T >> yy::Stack< T, S >::Stack () [inline]`

**11.78.2.2** `template<class T, class S = std::deque< T >> yy::Stack< T, S >::Stack (unsigned int n) [inline]`

## 11.78.3 Member Function Documentation

**11.78.3.1** `template<class T, class S = std::deque< T >> ConstIterator yy::Stack< T, S >::begin () const [inline]`

**11.78.3.2** `template<class T, class S = std::deque< T >> ConstIterator yy::Stack< T, S >::end () const [inline]`

**11.78.3.3** `template<class T, class S = std::deque< T >> unsigned int yy::Stack< T, S >::height () const [inline]`

**11.78.3.4** `] template<class T, class S = std::deque< T >> const T& yy::Stack< T, S >::operator[] (unsigned int i) const [inline]`

**11.78.3.5** `] template<class T, class S = std::deque< T >> T& yy::Stack< T, S >::operator[] (unsigned int i) [inline]`

**11.78.3.6** `template<class T, class S = std::deque< T >> void yy::Stack< T, S >::pop (unsigned int n = 1) [inline]`

**11.78.3.7** `template<class T, class S = std::deque< T >> void yy::Stack< T, S >::push (const T & t) [inline]`

## 11.78.4 Member Data Documentation

**11.78.4.1** `template<class T, class S = std::deque< T >> S yy::Stack< T, S >::seq_ [private]`

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

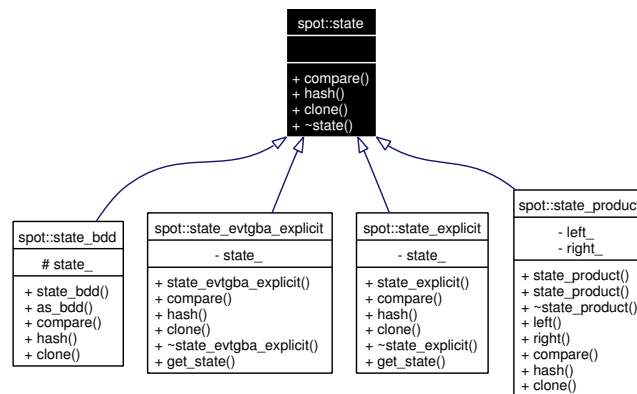
- [Itlparse/stack.hh](#)

## 11.79 spot::state Class Reference

Abstract class for states.

`#include <tgba/state.hh>`

Inheritance diagram for spot::state:



## Public Member Functions

- virtual int `compare` (const `state` \*other) const =0  
*Compares two states (that come from the same automaton).*
- virtual size\_t `hash` () const =0  
*Hash a state.*
- virtual `state` \* `clone` () const =0  
*Duplicate a state.*
- virtual `~state` ()

### 11.79.1 Detailed Description

Abstract class for states.

### 11.79.2 Constructor & Destructor Documentation

**11.79.2.1** virtual `spot::state::~~state` () [inline, virtual]

### 11.79.3 Member Function Documentation

**11.79.3.1** virtual `state`\* `spot::state::clone` () const [pure virtual]

Duplicate a state.

Implemented in `spot::state_evtgba_explicit`, `spot::state_bdd`, `spot::state_explicit`, and `spot::state_product`.

**11.79.3.2** virtual int `spot::state::compare` (const `state` \*other) const [pure virtual]

Compares two states (that come from the same automaton).

This method returns an integer less than, equal to, or greater than zero if *this* is found, respectively, to be less than, equal to, or greater than *other* according to some implicit total order.

This method should not be called to compare states from different automata.

See also:

[spot::state\\_ptr\\_less\\_than](#)

Implemented in [spot::state\\_bdd](#), and [spot::state\\_product](#).

### 11.79.3.3 virtual size\_t spot::state::hash() const [pure virtual]

Hash a state.

This method returns an integer that can be used as a hash value for this state.

Note that the hash value is guaranteed to be unique for all equal states (in [compare\(\)](#)'s sense) for only has long has one of these states exists. So it's OK to use a [spot::state](#) as a key in a [hash\\_map](#) because the mere use of the state as a key in the hash will ensure the state continues to exist.

However if you create the state, get its hash key, delete the state, recreate the same state, and get its hash key, you may obtain two different hash keys if the same state were not already used elsewhere. In practice this weird situation can occur only when the state is BDD-encoded, because BDD numbers (used to build the hash value) can be reused for other formulas. That probably doesn't matter, since the hash value is meant to be used in a [hash\\_map](#), but it had to be noted.

Implemented in [spot::state\\_evtgba\\_explicit](#), [spot::state\\_bdd](#), [spot::state\\_explicit](#), and [spot::state\\_product](#).

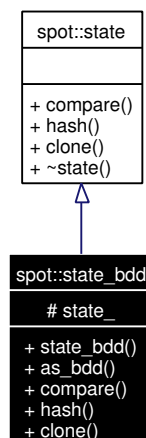
The documentation for this class was generated from the following file:

- [tgba/state.hh](#)

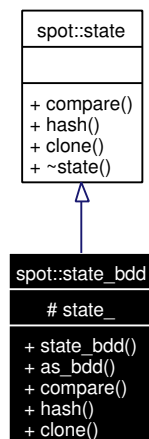
## 11.80 spot::state\_bdd Class Reference

```
#include <tgba/statebdd.hh>
```

Inheritance diagram for [spot::state\\_bdd](#):



Collaboration diagram for [spot::state\\_bdd](#):



### Public Member Functions

- `state_bdd` (bdd s)
- virtual bdd `as_bdd` () const  
*Return the BDD part of the state.*
- virtual int `compare` (const `state` \*other) const  
*Compares two states (that come from the same automaton).*
- virtual size\_t `hash` () const  
*Hash a state.*
- virtual `state_bdd` \* `clone` () const  
*Duplicate a state.*

### Protected Attributes

- bdd `state_`  
*BDD representation of the state.*

#### 11.80.1 Detailed Description

A state whose representation is a BDD.

#### 11.80.2 Constructor & Destructor Documentation

##### 11.80.2.1 `spot::state_bdd::state_bdd (bdd s)` [inline]



### 11.80.3 Member Function Documentation

#### 11.80.3.1 virtual bdd spot::state\_bdd::as\_bdd () const [inline, virtual]

Return the BDD part of the state.

#### 11.80.3.2 virtual state\_bdd\* spot::state\_bdd::clone () const [virtual]

Duplicate a state.

Implements [spot::state](#).

#### 11.80.3.3 virtual int spot::state\_bdd::compare (const state \* other) const [virtual]

Compares two states (that come from the same automaton).

This method returns an integer less than, equal to, or greater than zero if *this* is found, respectively, to be less than, equal to, or greater than *other* according to some implicit total order.

This method should not be called to compare states from different automata.

See also:

[spot::state\\_ptr\\_less\\_than](#)

Implements [spot::state](#).

#### 11.80.3.4 virtual size\_t spot::state\_bdd::hash () const [virtual]

Hash a state.

This method returns an integer that can be used as a hash value for this state.

Note that the hash value is guaranteed to be unique for all equal states (in [compare\(\)](#)'s sense) for only has long has one of these states exists. So it's OK to use a [spot::state](#) as a key in a `hash_map` because the mere use of the state as a key in the hash will ensure the state continues to exist.

However if you create the state, get its hash key, delete the state, recreate the same state, and get its hash key, you may obtain two different hash keys if the same state were not already used elsewhere. In practice this weird situation can occur only when the state is BDD-encoded, because BDD numbers (used to build the hash value) can be reused for other formulas. That probably doesn't matter, since the hash value is meant to be used in a `hash_map`, but it had to be noted.

Implements [spot::state](#).

### 11.80.4 Member Data Documentation

#### 11.80.4.1 bdd spot::state\_bdd::state\_ [protected]

BDD representation of the state.

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

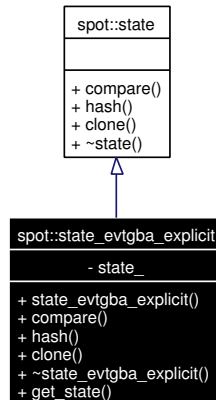
- [tgba/statebdd.hh](#)

## 11.81 spot::state\_evtgba\_explicit Class Reference

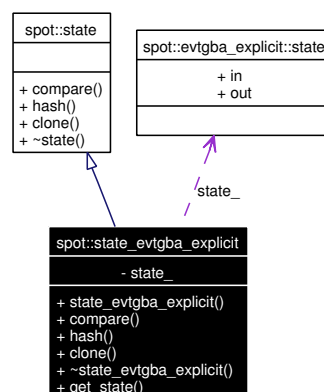
States used by `spot::tgba_evtgba_explicit`.

```
#include <evtgba/explicit.hh>
```

Inheritance diagram for spot::state\_evtgba\_explicit:



Collaboration diagram for spot::state\_evtgba\_explicit:



## Public Member Functions

- `state_evtgba_explicit` (const `evtgba_explicit::state` \*s)
- virtual int `compare` (const `spot::state` \*other) const
- virtual size\_t `hash` () const  
*Hash a state.*
- virtual `state_evtgba_explicit` \* `clone` () const  
*Duplicate a state.*
- virtual `~state_evtgba_explicit` ()
- const `evtgba_explicit::state` \* `get_state` () const
- virtual int `compare` (const `state` \*other) const =0  
*Compares two states (that come from the same automaton).*

## Private Attributes

- const `evtgba_explicit::state` \* `state_`

### 11.81.1 Detailed Description

States used by `spot::tgba_evtgba_explicit`.

### 11.81.2 Constructor & Destructor Documentation

**11.81.2.1** `spot::state_evtgba_explicit::state_evtgba_explicit (const evtgba_explicit::state * s)` `[inline]`

**11.81.2.2** `virtual spot::state_evtgba_explicit::~state_evtgba_explicit ()` `[inline, virtual]`

### 11.81.3 Member Function Documentation

**11.81.3.1** `virtual state_evtgba_explicit* spot::state_evtgba_explicit::clone () const` `[virtual]`

Duplicate a state.

Implements `spot::state`.

**11.81.3.2** `virtual int spot::state::compare (const state * other) const` `[pure virtual, inherited]`

Compares two states (that come from the same automaton).

This method returns an integer less than, equal to, or greater than zero if *this* is found, respectively, to be less than, equal to, or greater than *other* according to some implicit total order.

This method should not be called to compare states from different automata.

See also:

`spot::state_ptr_less_than`

Implemented in `spot::state_bdd`, and `spot::state_product`.

**11.81.3.3** `virtual int spot::state_evtgba_explicit::compare (const spot::state * other) const` `[virtual]`

**11.81.3.4** `const evtgba_explicit::state* spot::state_evtgba_explicit::get_state () const`

**11.81.3.5** `virtual size_t spot::state_evtgba_explicit::hash () const` `[virtual]`

Hash a state.

This method returns an integer that can be used as a hash value for this state.

Note that the hash value is guaranteed to be unique for all equal states (in `compare()`'s sense) for only has long has one of these states exists. So it's OK to use a `spot::state` as a key in a `hash_map` because the mere use of the state as a key in the hash will ensure the state continues to exist.

However if you create the state, get its hash key, delete the state, recreate the same state, and get its hash key, you may obtain two different hash keys if the same state were not already used elsewhere. In practice this weird situation can occur only when the state is BDD-encoded, because BDD numbers (used to build the hash value) can be reused for other formulas. That probably doesn't matter, since the hash value is meant to be used in a `hash_map`, but it had to be noted.

Implements [spot::state](#).

#### 11.81.4 Member Data Documentation

##### 11.81.4.1 `const evtgba_explicit::state* spot::state_evtgba_explicit::state_ [private]`

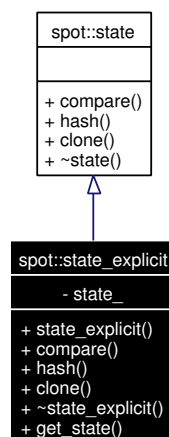
The documentation for this class was generated from the following file:

- [evtgba/explicit.hh](#)

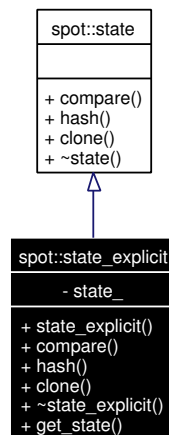
## 11.82 spot::state\_explicit Class Reference

```
#include <tgba/tgbaexplicit.hh>
```

Inheritance diagram for `spot::state_explicit`:



Collaboration diagram for `spot::state_explicit`:



### Public Member Functions

- `state_explicit` (const `tgba_explicit::state` \*s)
- virtual int `compare` (const `spot::state` \*other) const
- virtual size\_t `hash` () const  
*Hash a state.*
- virtual `state_explicit` \* `clone` () const  
*Duplicate a state.*
- virtual `~state_explicit` ()
- const `tgba_explicit::state` \* `get_state` () const
- virtual int `compare` (const `state` \*other) const =0  
*Compares two states (that come from the same automaton).*

### Private Attributes

- const `tgba_explicit::state` \* `state_`

#### 11.82.1 Detailed Description

States used by `spot::tgba_explicit`.

#### 11.82.2 Constructor & Destructor Documentation

**11.82.2.1** `spot::state_explicit::state_explicit` (const `tgba_explicit::state` \*s) [inline]

**11.82.2.2** virtual `spot::state_explicit::~~state_explicit` () [inline, virtual]

### 11.82.3 Member Function Documentation

#### 11.82.3.1 `virtual state\_explicit* spot::state_explicit::clone () const` [virtual]

Duplicate a state.

Implements [spot::state](#).

#### 11.82.3.2 `virtual int spot::state::compare (const state * other) const` [pure virtual, inherited]

Compares two states (that come from the same automaton).

This method returns an integer less than, equal to, or greater than zero if *this* is found, respectively, to be less than, equal to, or greater than *other* according to some implicit total order.

This method should not be called to compare states from different automata.

See also:

[spot::state\\_ptr\\_less\\_than](#)

Implemented in [spot::state\\_bdd](#), and [spot::state\\_product](#).

#### 11.82.3.3 `virtual int spot::state_explicit::compare (const spot::state * other) const` [virtual]

#### 11.82.3.4 `const tgba\_explicit::state* spot::state_explicit::get_state () const`

#### 11.82.3.5 `virtual size_t spot::state_explicit::hash () const` [virtual]

Hash a state.

This method returns an integer that can be used as a hash value for this state.

Note that the hash value is guaranteed to be unique for all equal states (in [compare\(\)](#)'s sense) for only has long has one of these states exists. So it's OK to use a [spot::state](#) as a key in a `hash_map` because the mere use of the state as a key in the hash will ensure the state continues to exist.

However if you create the state, get its hash key, delete the state, recreate the same state, and get its hash key, you may obtain two different hash keys if the same state were not already used elsewhere. In practice this weird situation can occur only when the state is BDD-encoded, because BDD numbers (used to build the hash value) can be reused for other formulas. That probably doesn't matter, since the hash value is meant to be used in a `hash_map`, but it had to be noted.

Implements [spot::state](#).

### 11.82.4 Member Data Documentation

#### 11.82.4.1 `const tgba\_explicit::state* spot::state_explicit::state_` [private]

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

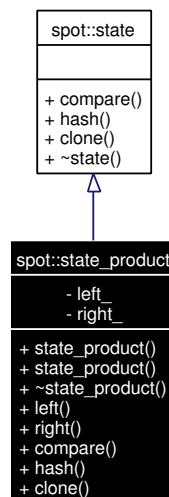
- [tgba/tgbaexplicit.hh](#)

## 11.83 spot::state\_product Class Reference

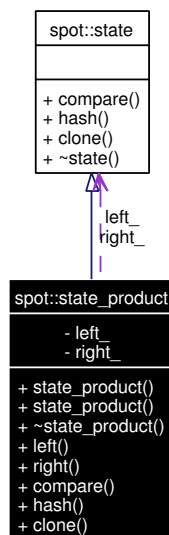
A state for [spot::tgba\\_product](#).

```
#include <tgba/tgbaproduct.hh>
```

Inheritance diagram for spot::state\_product:



Collaboration diagram for spot::state\_product:



### Public Member Functions

- `state_product` (`state *left`, `state *right`)

*Constructor.*

- `state_product` (const `state_product` &o)  
*Copy constructor.*
- virtual `~state_product` ()
- `state * left` () const
- `state * right` () const
- virtual int `compare` (const `state` \*other) const  
*Compares two states (that come from the same automaton).*
- virtual size\_t `hash` () const  
*Hash a state.*
- virtual `state_product * clone` () const  
*Duplicate a state.*

### Private Attributes

- `state * left_`  
*State from the left automaton.*
- `state * right_`  
*State from the right automaton.*

### 11.83.1 Detailed Description

A state for `spot::tgba_product`.

This state is in fact a pair of state: the state from the left automaton and that of the right.

### 11.83.2 Constructor & Destructor Documentation

#### 11.83.2.1 `spot::state_product::state_product (state * left, state * right)` [inline]

Constructor.

#### Parameters:

*left* The state from the left automaton.

*right* The state from the right automaton. These states are acquired by `spot::state_product`, and will be deleted on destruction.

#### 11.83.2.2 `spot::state_product::state_product (const state_product & o)`

Copy constructor.

#### 11.83.2.3 `virtual spot::state_product::~~state_product ()` [virtual]



### 11.83.3 Member Function Documentation

#### 11.83.3.1 `virtual state_product* spot::state_product::clone () const` [virtual]

Duplicate a state.

Implements [spot::state](#).

#### 11.83.3.2 `virtual int spot::state_product::compare (const state * other) const` [virtual]

Compares two states (that come from the same automaton).

This method returns an integer less than, equal to, or greater than zero if *this* is found, respectively, to be less than, equal to, or greater than *other* according to some implicit total order.

This method should not be called to compare states from different automata.

See also:

[spot::state\\_ptr\\_less\\_than](#)

Implements [spot::state](#).

#### 11.83.3.3 `virtual size_t spot::state_product::hash () const` [virtual]

Hash a state.

This method returns an integer that can be used as a hash value for this state.

Note that the hash value is guaranteed to be unique for all equal states (in [compare\(\)](#)'s sense) for only has long has one of these states exists. So it's OK to use a [spot::state](#) as a key in a `hash_map` because the mere use of the state as a key in the hash will ensure the state continues to exist.

However if you create the state, get its hash key, delete the state, recreate the same state, and get its hash key, you may obtain two different hash keys if the same state were not already used elsewhere. In practice this weird situation can occur only when the state is BDD-encoded, because BDD numbers (used to build the hash value) can be reused for other formulas. That probably doesn't matter, since the hash value is meant to be used in a `hash_map`, but it had to be noted.

Implements [spot::state](#).

#### 11.83.3.4 `state* spot::state_product::left () const` [inline]

#### 11.83.3.5 `state* spot::state_product::right () const` [inline]

### 11.83.4 Member Data Documentation

#### 11.83.4.1 `state* spot::state_product::left_` [private]

State from the left automaton.

#### 11.83.4.2 `state* spot::state_product::right_` [private]

State from the right automaton.

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

- [tgba/tgbaproduct.hh](#)

## 11.84 spot::state\_ptr\_equal Struct Reference

An Equivalence Relation for state\*.

```
#include <tgba/state.hh>
```

### Public Member Functions

- bool [operator\(\)](#) (const [state](#) \*left, const [state](#) \*right) const

#### 11.84.1 Detailed Description

An Equivalence Relation for state\*.

This is meant to be used as a comparison functor for Sgi hash\_map whose key are of type state\*.

For instance here is how one could declare a map of state\*.

```
// Remember how many times each state has been visited.
Sgi::hash_map<spot::state*, int, spot::state_ptr_hash,
              spot::state_ptr_equal> seen;
```

#### 11.84.2 Member Function Documentation

**11.84.2.1** bool [spot::state\\_ptr\\_equal::operator\(\)](#) (const [state](#) \* *left*, const [state](#) \* *right*) const  
[inline]

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

- [tgba/state.hh](#)

## 11.85 spot::state\_ptr\_hash Struct Reference

Hash Function for state\*.

```
#include <tgba/state.hh>
```

### Public Member Functions

- size\_t [operator\(\)](#) (const [state](#) \*that) const

#### 11.85.1 Detailed Description

Hash Function for state\*.

This is meant to be used as a hash functor for Sgi's hash\_map whose key are of type state\*.

For instance here is how one could declare a map of state\*.

```
// Remember how many times each state has been visited.
Sgi::hash_map<spot::state*, int, spot::state_ptr_hash,
              spot::state_ptr_equal> seen;
```

## 11.85.2 Member Function Documentation

### 11.85.2.1 size\_t spot::state\_ptr\_hash::operator() (const state \* *that*) const [inline]

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

- [tgba/state.hh](#)

## 11.86 spot::state\_ptr\_less\_than Struct Reference

Strict Weak Ordering for state\*.

```
#include <tgba/state.hh>
```

### Public Member Functions

- bool [operator\(\)](#) (const state \*left, const state \*right) const

### 11.86.1 Detailed Description

Strict Weak Ordering for state\*.

This is meant to be used as a comparison functor for STL map whose key are of type state\*.

For instance here is how one could declare a map of state\*.

```
// Remember how many times each state has been visited.
std::map<spot::state*, int, spot::state_ptr_less_than> seen;
```

## 11.86.2 Member Function Documentation

### 11.86.2.1 bool spot::state\_ptr\_less\_than::operator() (const state \* *left*, const state \* *right*) const [inline]

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

- [tgba/state.hh](#)

## 11.87 spot::string\_hash Struct Reference

A hash function for strings.

```
#include <misc/hash.hh>
```

### Public Member Functions

- size\_t [operator\(\)](#) (const std::string &s) const

### 11.87.1 Detailed Description

A hash function for strings.

## 11.87.2 Member Function Documentation

### 11.87.2.1 size\_t spot::string\_hash::operator() (const std::string &s) const [inline]

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

- misc/[hash.hh](#)

## 11.88 spot::symbol Class Reference

```
#include <evtgba/symbol.hh>
```

### Public Member Functions

- const std::string & [name](#) () const
- void [ref](#) () const
- void [unref](#) () const

### Static Public Member Functions

- static const [symbol](#) \* [instance](#) (const std::string &name)
- static unsigned [instance\\_count](#) ()  
*Number of instantiated atomic propositions. For debugging.*
- static std::ostream & [dump\\_instances](#) (std::ostream &os)  
*List all instances of atomic propositions. For debugging.*

### Protected Types

- typedef std::map< const std::string, const [symbol](#) \* > [map](#)

### Protected Member Functions

- int [ref\\_count](#) () const
- [symbol](#) (const std::string \*name)
- [~symbol](#) ()

### Static Protected Attributes

- static [map](#) [instances\\_](#)

### Private Member Functions

- [symbol](#) (const [symbol](#) &)

**Private Attributes**

- const std::string \* [name\\_](#)  
*Undefined.*
- int [refs\\_](#)

**11.88.1 Member Typedef Documentation**

**11.88.1.1** typedef std::map<const std::string, const [symbol\\*](#)> [spot::symbol::map](#) [protected]

**11.88.2 Constructor & Destructor Documentation**

**11.88.2.1** spot::symbol::symbol (const std::string \* *name*) [protected]

**11.88.2.2** spot::symbol::~~symbol () [protected]

**11.88.2.3** spot::symbol::symbol (const [symbol](#) &) [private]

**11.88.3 Member Function Documentation**

**11.88.3.1** static std::ostream& spot::symbol::dump\_instances (std::ostream & *os*) [static]

List all instances of atomic propositions. For debugging.

**11.88.3.2** static const [symbol\\*](#) spot::symbol::instance (const std::string & *name*) [static]

**11.88.3.3** static unsigned spot::symbol::instance\_count () [static]

Number of instantiated atomic propositions. For debugging.

**11.88.3.4** const std::string& spot::symbol::name () const

**11.88.3.5** void spot::symbol::ref () const

**11.88.3.6** int spot::symbol::ref\_count\_ () const [protected]

**11.88.3.7** void spot::symbol::unref () const

**11.88.4 Member Data Documentation**

**11.88.4.1** [map](#) [spot::symbol::instances\\_](#) [static, protected]

**11.88.4.2** const std::string\* [spot::symbol::name\\_](#) [private]

Undefined.

### 11.88.4.3 int spot::symbol::refs\_ [mutable, private]

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

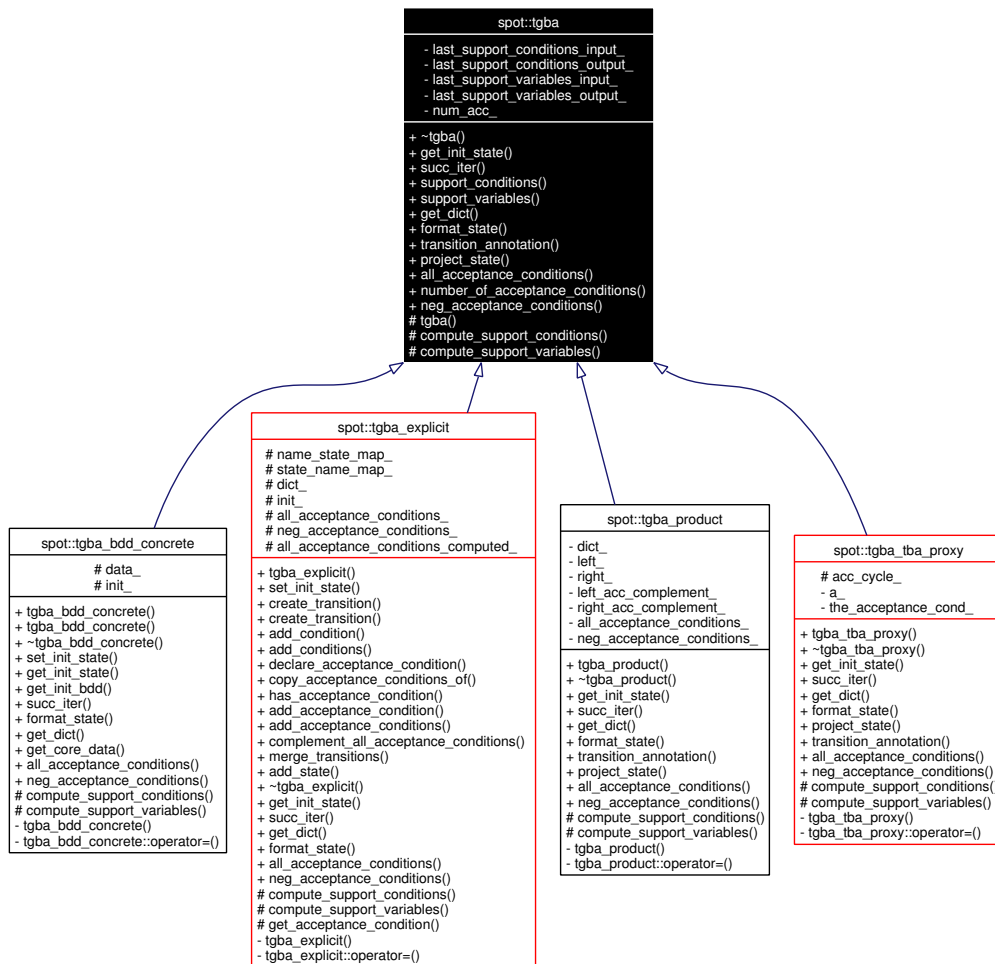
- [evtgba/symbol.hh](#)

## 11.89 spot::tgba Class Reference

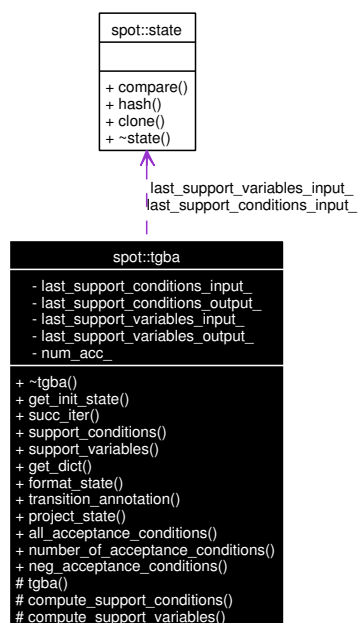
A Transition-based Generalized Büchi Automaton.

```
#include <tgba/tgba.hh>
```

Inheritance diagram for spot::tgba:



Collaboration diagram for spot::tgba:



## Public Member Functions

- virtual `~tgba()`
- virtual `state * get_init_state()` const =0  
*Get the initial state of the automaton.*
- virtual `tgba_succ_iterator * succ_iter` (const `state *local_state`, const `state *global_state=0`, const `tgba *global_automaton=0`) const =0  
*Get an iterator over the successors of local\_state.*
- bdd `support_conditions` (const `state *state`) const  
*Get a formula that must hold whatever successor is taken.*
- bdd `support_variables` (const `state *state`) const  
*Get the conjunctions of variables tested by the outgoing transitions of state.*
- virtual bdd `dict * get_dict()` const =0  
*Get the dictionary associated to the automaton.*
- virtual std::string `format_state` (const `state *state`) const =0  
*Format the state as a string for printing.*
- virtual std::string `transition_annotation` (const `tgba_succ_iterator *t`) const  
*Return a possible annotation for the transition pointed to by the iterator.*
- virtual `state * project_state` (const `state *s`, const `tgba *t`) const  
*Project a state on an automaton.*
- virtual bdd `all_acceptance_conditions()` const =0

*Return the set of all acceptance conditions used by this automaton.*

- virtual int [number\\_of\\_acceptance\\_conditions](#) () const  
*The number of acceptance conditions.*
- virtual bdd [neg\\_acceptance\\_conditions](#) () const =0  
*Return the conjunction of all negated acceptance variables.*

### Protected Member Functions

- [tgba](#) ()
- virtual bdd [compute\\_support\\_conditions](#) (const [state](#) \*state) const =0  
*Do the actual computation of [tgba::support\\_conditions\(\)](#).*
- virtual bdd [compute\\_support\\_variables](#) (const [state](#) \*state) const =0  
*Do the actual computation of [tgba::support\\_variables\(\)](#).*

### Private Attributes

- const [state](#) \* [last\\_support\\_conditions\\_input\\_](#)
- bdd [last\\_support\\_conditions\\_output\\_](#)
- const [state](#) \* [last\\_support\\_variables\\_input\\_](#)
- bdd [last\\_support\\_variables\\_output\\_](#)
- int [num\\_acc\\_](#)

## 11.89.1 Detailed Description

A Transition-based Generalized Büchi Automaton.

The acronym TGBA (Transition-based Generalized Büchi Automaton) was coined by Dimitra Gianakopoulou and Flavio Lerda in "From States to Transitions: Improving Translation of LTL Formulae to Büchi Automata". (FORTE'02)

TGBAs are transition-based, meanings their labels are put on arcs, not on nodes. They use Generalized Büchi acceptance conditions: there are several acceptance sets (of transitions), and a path can be accepted only if it traverse at least one transition of each set infinitely often.

Browsing such automaton can be achieved using two functions. `get_init_state`, and `succ_iter`. The former returns the initial state while the latter allows to explore the successor states of any state.

Note that although this is a transition-based automata, we never represent transitions! Transition informations are obtained by querying the iterator over the successors of a state.

## 11.89.2 Constructor & Destructor Documentation

### 11.89.2.1 spot::tgba::tgba () [protected]

### 11.89.2.2 virtual spot::tgba::~~tgba () [virtual]



### 11.89.3 Member Function Documentation

#### 11.89.3.1 virtual bdd spot::tgba::all\_acceptance\_conditions () const [pure virtual]

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implemented in [spot::tgba\\_bdd\\_concrete](#), [spot::tgba\\_explicit](#), [spot::tgba\\_product](#), and [spot::tgba\\_tba\\_proxy](#).

#### 11.89.3.2 virtual bdd spot::tgba::compute\_support\_conditions (const state \* state) const [protected, pure virtual]

Do the actual computation of [tgba::support\\_conditions\(\)](#).

Implemented in [spot::tgba\\_bdd\\_concrete](#), [spot::tgba\\_product](#), and [spot::tgba\\_tba\\_proxy](#).

#### 11.89.3.3 virtual bdd spot::tgba::compute\_support\_variables (const state \* state) const [protected, pure virtual]

Do the actual computation of [tgba::support\\_variables\(\)](#).

Implemented in [spot::tgba\\_bdd\\_concrete](#), [spot::tgba\\_product](#), and [spot::tgba\\_tba\\_proxy](#).

#### 11.89.3.4 virtual std::string spot::tgba::format\_state (const state \* state) const [pure virtual]

Format the state as a string for printing.

This formatting is the responsibility of the automata who owns the state.

Implemented in [spot::tgba\\_bdd\\_concrete](#), [spot::tgba\\_product](#), and [spot::tgba\\_tba\\_proxy](#).

#### 11.89.3.5 virtual bdd\_dict\* spot::tgba::get\_dict () const [pure virtual]

Get the dictionary associated to the automaton.

State are represented as BDDs. The dictionary allows to map BDD variables back to formulae, and vice versa. This is useful when dealing with several automata (which may use the same BDD variable for different formula), or simply when printing.

Implemented in [spot::tgba\\_bdd\\_concrete](#), [spot::tgba\\_explicit](#), [spot::tgba\\_product](#), and [spot::tgba\\_tba\\_proxy](#).

#### 11.89.3.6 virtual state\* spot::tgba::get\_init\_state () const [pure virtual]

Get the initial state of the automaton.

The state has been allocated with new. It is the responsibility of the caller to delete it when no longer needed.

Implemented in [spot::tgba\\_bdd\\_concrete](#), [spot::tgba\\_explicit](#), [spot::tgba\\_product](#), and [spot::tgba\\_tba\\_proxy](#).

#### 11.89.3.7 virtual bdd spot::tgba::neg\_acceptance\_conditions () const [pure virtual]

Return the conjunction of all negated acceptance variables.

For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a]&!Acc[b]&!Acc[c]`.

This is useful when making products: each operand's condition set should be augmented with the [neg\\_acceptance\\_conditions\(\)](#) of the other operand.

Implemented in [spot::tgba\\_bdd\\_concrete](#), [spot::tgba\\_explicit](#), [spot::tgba\\_product](#), and [spot::tgba\\_tba\\_proxy](#).

#### 11.89.3.8 virtual int spot::tgba::number\_of\_acceptance\_conditions () const [virtual]

The number of acceptance conditions.

#### 11.89.3.9 virtual state\* spot::tgba::project\_state (const state \*s, const tgba \*t) const [virtual]

Project a state on an automaton.

This converts *s*, into that corresponding [spot::state](#) for *t*. This is useful when you have the state of a product, and want restrict this state to a specific automata occuring in the product.

It goes without saying that *s* and *t* should be compatible (i.e., *s* is a state of *t*).

##### Returns:

0 if the projection fails (*s* is unrelated to *t*), or a new `state*` (the projected state) that must be deleted by the caller.

Reimplemented in [spot::tgba\\_product](#), and [spot::tgba\\_tba\\_proxy](#).

#### 11.89.3.10 virtual tgba\_succ\_iterator\* spot::tgba::succ\_iter (const state \*local\_state, const state \*global\_state = 0, const tgba \*global\_automaton = 0) const [pure virtual]

Get an iterator over the successors of *local\_state*.

The iterator has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

During synchornized products, additional informations are passed about the entire product and its state. Recall that products can be nested, forming a tree of `spot::tgba` where most values are computed on demand. *global\_automaton* designate the root `spot::tgba`, and *global\_state* its state. This two objects can be used by [succ\\_iter\(\)](#) to restrict the set of successors to compute.

##### Parameters:

**local\_state** The state whose successors are to be explored. This pointer is not adopted in any way by `succ_iter`, and it is still the caller's responsibility to delete it when appropriate (this can be done during the lifetime of the iterator).

**global\_state** In a product, the state of the global product automaton. Otherwise, 0. Like *locale\_state*, *global\_state* is not adopted by `succ_iter`.

**global\_automaton** In a product, the global product automaton. Otherwise, 0.

Implemented in [spot::tgba\\_bdd\\_concrete](#), [spot::tgba\\_product](#), and [spot::tgba\\_tba\\_proxy](#).

**11.89.3.11 bdd spot::tgba::support\_conditions (const [state](#) \* *state*) const**

Get a formula that must hold whatever successor is taken.

**Returns:**

A formula which must be verified for all successors of *state*.

This can be as simple as `bddtrue`, or more completely the disjunction of the condition of all successors. This is used as an hint by `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_conditions()`, this function is just a wrapper that will cache the last return value for efficiency.

**11.89.3.12 bdd spot::tgba::support\_variables (const [state](#) \* *state*) const**

Get the conjunctions of variables tested by the outgoing transitions of *state*.

All variables tested by outgoing transitions must be returned. This is mandatory.

This is used as an hint by some `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_variables()`, this function is just a wrapper that will cache the last return value for efficiency.

**11.89.3.13 virtual std::string spot::tgba::transition\_annotation (const [tgba\\_succ\\_iterator](#) \* *t*) const [virtual]**

Return a possible annotation for the transition pointed to by the iterator.

Implementing this function is optional; the default annotation is the empty string.

**Parameters:**

*t* a non-done [tgba\\_succ\\_iterator](#) for this automata

Reimplemented in `spot::tgba_product`, and `spot::tgba_tba_proxy`.

**11.89.4 Member Data Documentation**

**11.89.4.1** `const state* spot::tgba::last\_support\_conditions\_input\_` [mutable, private]

**11.89.4.2** `bdd spot::tgba::last\_support\_conditions\_output\_` [mutable, private]

**11.89.4.3** `const state* spot::tgba::last\_support\_variables\_input\_` [mutable, private]

**11.89.4.4** `bdd spot::tgba::last\_support\_variables\_output\_` [mutable, private]

**11.89.4.5** `int spot::tgba::num\_acc\_` [mutable, private]

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

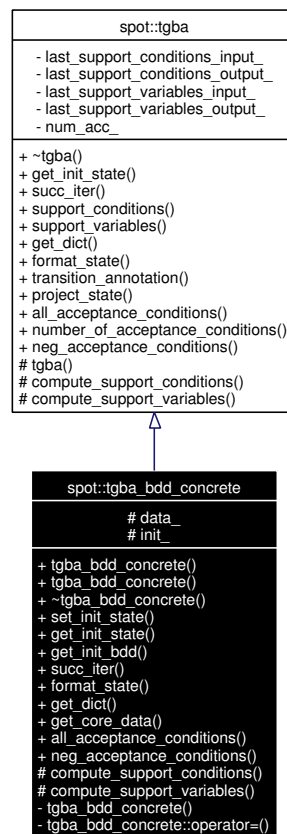
- [tgba/tgba.hh](#)

## 11.90 spot::tgba\_bdd\_concrete Class Reference

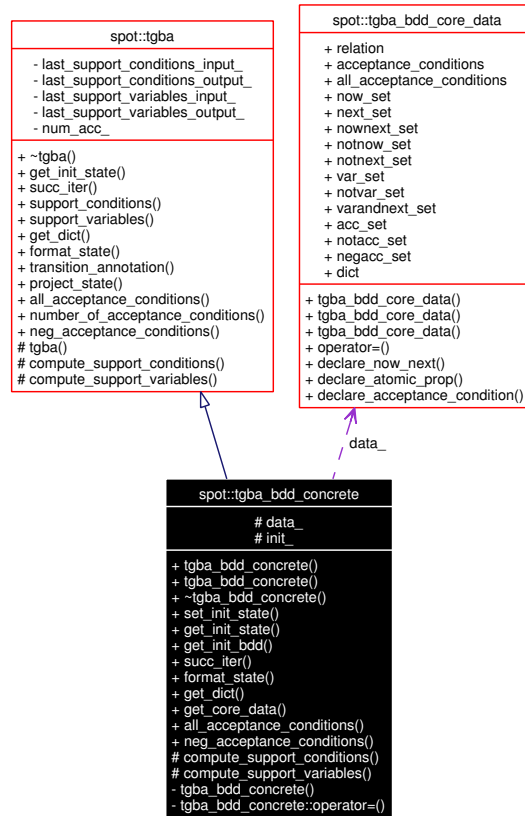
A concrete [spot::tgba](#) implemented using BDDs.

```
#include <tgba/tgbabddconcrete.hh>
```

Inheritance diagram for spot::tgba\_bdd\_concrete:



Collaboration diagram for spot::tgba\_bdd\_concrete:



## Public Member Functions

- [tgba\\_bdd\\_concrete](#) (const [tgba\\_bdd\\_factory](#) &fact)  
*Construct a [tgba\\_bdd\\_concrete](#) with unknown initial state.*
- [tgba\\_bdd\\_concrete](#) (const [tgba\\_bdd\\_factory](#) &fact, bdd init)  
*Construct a [tgba\\_bdd\\_concrete](#) with known initial state.*
- virtual [~tgba\\_bdd\\_concrete](#) ()
- virtual void [set\\_init\\_state](#) (bdd s)  
*Set the initial state.*
- virtual [state\\_bdd](#) \* [get\\_init\\_state](#) () const  
*Get the initial state of the automaton.*
- bdd [get\\_init\\_bdd](#) () const  
*Get the initial state directly as a BDD.*
- virtual [tgba\\_succ\\_iterator\\_concrete](#) \* [succ\\_iter](#) (const [state](#) \*local\_state, const [state](#) \*global\_state=0, const [tgba](#) \*global\_automaton=0) const  
*Get an iterator over the successors of local\_state.*
- virtual std::string [format\\_state](#) (const [state](#) \*state) const

*Format the state as a string for printing.*

- virtual `bdd_dict * get_dict ()` const  
*Get the dictionary associated to the automaton.*
- const `tgba_bdd_core_data & get_core_data ()` const  
*Get the core data associated to this automaton.*
- virtual `bdd all_acceptance_conditions ()` const  
*Return the set of all acceptance conditions used by this automaton.*
- virtual `bdd neg_acceptance_conditions ()` const  
*Return the conjunction of all negated acceptance variables.*
- `bdd support_conditions (const state *state)` const  
*Get a formula that must hold whatever successor is taken.*
- `bdd support_variables (const state *state)` const  
*Get the conjunctions of variables tested by the outgoing transitions of state.*
- virtual `std::string transition_annotation (const tgba_succ_iterator *t)` const  
*Return a possible annotation for the transition pointed to by the iterator.*
- virtual `state * project_state (const state *s, const tgba *t)` const  
*Project a state on an automaton.*
- virtual `int number_of_acceptance_conditions ()` const  
*The number of acceptance conditions.*

### Protected Member Functions

- virtual `bdd compute_support_conditions (const state *state)` const  
*Do the actual computation of `tgba::support_conditions()`.*
- virtual `bdd compute_support_variables (const state *state)` const  
*Do the actual computation of `tgba::support_variables()`.*

### Protected Attributes

- `tgba_bdd_core_data data_`  
*Core data associated to the automaton.*
- `bdd init_`  
*Initial state.*

## Private Member Functions

- [tgba\\_bdd\\_concrete](#) (const [tgba\\_bdd\\_concrete](#) &)
- [tgba\\_bdd\\_concrete](#) & [tgba\\_bdd\\_concrete::operator=](#) (const [tgba\\_bdd\\_concrete](#) &)

### 11.90.1 Detailed Description

A concrete [spot::tgba](#) implemented using BDDs.

### 11.90.2 Constructor & Destructor Documentation

#### 11.90.2.1 spot::tgba\_bdd\_concrete::tgba\_bdd\_concrete (const [tgba\\_bdd\\_factory](#) & *fact*)

Construct a [tgba\\_bdd\\_concrete](#) with unknown initial state.

[set\\_init\\_state\(\)](#) should be called later.

#### 11.90.2.2 spot::tgba\_bdd\_concrete::tgba\_bdd\_concrete (const [tgba\\_bdd\\_factory](#) & *fact*, bdd *init*)

Construct a [tgba\\_bdd\\_concrete](#) with known initial state.

#### 11.90.2.3 virtual spot::tgba\_bdd\_concrete::~~[tgba\\_bdd\\_concrete](#) () [virtual]

#### 11.90.2.4 spot::tgba\_bdd\_concrete::tgba\_bdd\_concrete (const [tgba\\_bdd\\_concrete](#) &) [private]

### 11.90.3 Member Function Documentation

#### 11.90.3.1 virtual bdd spot::tgba\_bdd\_concrete::all\_acceptance\_conditions () const [virtual]

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::tgba](#).

#### 11.90.3.2 virtual bdd spot::tgba\_bdd\_concrete::compute\_support\_conditions (const [state](#) \* *state*) const [protected, virtual]

Do the actual computation of [tgba::support\\_conditions\(\)](#).

Implements [spot::tgba](#).

#### 11.90.3.3 virtual bdd spot::tgba\_bdd\_concrete::compute\_support\_variables (const [state](#) \* *state*) const [protected, virtual]

Do the actual computation of [tgba::support\\_variables\(\)](#).

Implements [spot::tgba](#).

#### 11.90.3.4 virtual std::string spot::tgba\_bdd\_concrete::format\_state (const state \* state) const [virtual]

Format the state as a string for printing.

This formatting is the responsibility of the automata who owns the state.

Implements [spot::tgba](#).

#### 11.90.3.5 const tgba\_bdd\_core\_data& spot::tgba\_bdd\_concrete::get\_core\_data () const

Get the core data associated to this automaton.

These data includes the various BDD used to represent the relation, encode variable sets, Next-to-Now rewrite rules, etc.

#### 11.90.3.6 virtual bdd\_dict\* spot::tgba\_bdd\_concrete::get\_dict () const [virtual]

Get the dictionary associated to the automaton.

State are represented as BDDs. The dictionary allows to map BDD variables back to formulae, and vice versa. This is useful when dealing with several automata (which may use the same BDD variable for different formula), or simply when printing.

Implements [spot::tgba](#).

#### 11.90.3.7 bdd spot::tgba\_bdd\_concrete::get\_init\_bdd () const

Get the initial state directly as a BDD.

The sole point of this method is to prevent writing horrors such as

```
state_bdd* s = automata.get_init_state();
some_class some_instance(s->as_bdd());
delete s;
```

#### 11.90.3.8 virtual state\_bdd\* spot::tgba\_bdd\_concrete::get\_init\_state () const [virtual]

Get the initial state of the automaton.

The state has been allocated with new. It is the responsibility of the caller to delete it when no longer needed.

Implements [spot::tgba](#).

#### 11.90.3.9 virtual bdd spot::tgba\_bdd\_concrete::neg\_acceptance\_conditions () const [virtual]

Return the conjunction of all negated acceptance variables.

For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a]&!Acc[b]&!Acc[c]`.

This is useful when making products: each operand's condition set should be augmented with the [neg\\_acceptance\\_conditions\(\)](#) of the other operand.

Implements [spot::tgba](#).



**11.90.3.10** `virtual int spot::tgba::number_of_acceptance_conditions () const [virtual, inherited]`

The number of acceptance conditions.

**11.90.3.11** `virtual state* spot::tgba::project_state (const state * s, const tgba * t) const [virtual, inherited]`

Project a state on an automaton.

This converts *s*, into that corresponding `spot::state` for *t*. This is useful when you have the state of a product, and want restrict this state to a specific automata occuring in the product.

It goes without saying that *s* and *t* should be compatible (i.e., *s* is a state of *t*).

**Returns:**

0 if the projection fails (*s* is unrelated to *t*), or a new `state*` (the projected state) that must be deleted by the caller.

Reimplemented in `spot::tgba_product`, and `spot::tgba_tba_proxy`.

**11.90.3.12** `virtual void spot::tgba_bdd_concrete::set_init_state (bdd s) [virtual]`

Set the initial state.

**11.90.3.13** `virtual tgba_succ_iterator_concrete* spot::tgba_bdd_concrete::succ_iter (const state * local_state, const state * global_state = 0, const tgba * global_automaton = 0) const [virtual]`

Get an iterator over the successors of *local\_state*.

The iterator has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

During synchornized products, additional informations are passed about the entire product and its state. Recall that products can be nested, forming a tree of `spot::tgba` where most values are computed on demand. *global\_automaton* designate the root `spot::tgba`, and *global\_state* its state. This two objects can be used by `succ_iter()` to restrict the set of successors to compute.

**Parameters:**

*local\_state* The state whose successors are to be explored. This pointer is not adopted in any way by `succ_iter`, and it is still the caller's responsibility to delete it when appropriate (this can be done during the lifetime of the iterator).

*global\_state* In a product, the state of the global product automaton. Otherwise, 0. Like *locale\_state*, *global\_state* is not adopted by `succ_iter`.

*global\_automaton* In a product, the global product automaton. Otherwise, 0.

Implements `spot::tgba`.

**11.90.3.14** `bdd spot::tgba::support_conditions (const state * state) const [inherited]`

Get a formula that must hold whatever successor is taken.

**Returns:**

A formula which must be verified for all successors of *state*.

This can be as simple as `bddtrue`, or more completely the disjunction of the condition of all successors. This is used as an hint by `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_conditions()`, this function is just a wrapper that will cache the last return value for efficiency.

#### 11.90.3.15 `bdd spot::tgba::support_variables (const state * state) const` [inherited]

Get the conjunctions of variables tested by the outgoing transitions of *state*.

All variables tested by outgoing transitions must be returned. This is mandatory.

This is used as an hint by some `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_variables()`, this function is just a wrapper that will cache the last return value for efficiency.

#### 11.90.3.16 `tgba\_bdd\_concrete& spot::tgba_bdd_concrete::tgba_bdd_concrete::operator= (const tgba\_bdd\_concrete &) [private]`

#### 11.90.3.17 `virtual std::string spot::tgba::transition_annotation (const tgba\_succ\_iterator * t) const` [virtual, inherited]

Return a possible annotation for the transition pointed to by the iterator.

Implementing this function is optional; the default annotation is the empty string.

#### Parameters:

*t* a non-done [tgba\\_succ\\_iterator](#) for this automata

Reimplemented in `spot::tgba_product`, and `spot::tgba_tba_proxy`.

### 11.90.4 Member Data Documentation

#### 11.90.4.1 `tgba\_bdd\_core\_data spot::tgba_bdd_concrete::data_` [protected]

Core data associated to the automaton.

#### 11.90.4.2 `bdd spot::tgba_bdd_concrete::init_` [protected]

Initial state.

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

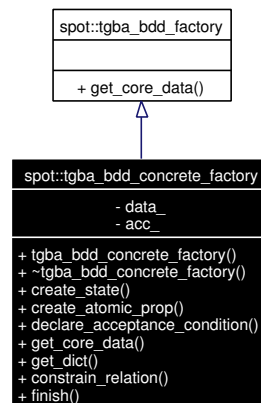
- [tgba/tgabddconcrete.hh](#)

## 11.91 `spot::tgba_bdd_concrete_factory` Class Reference

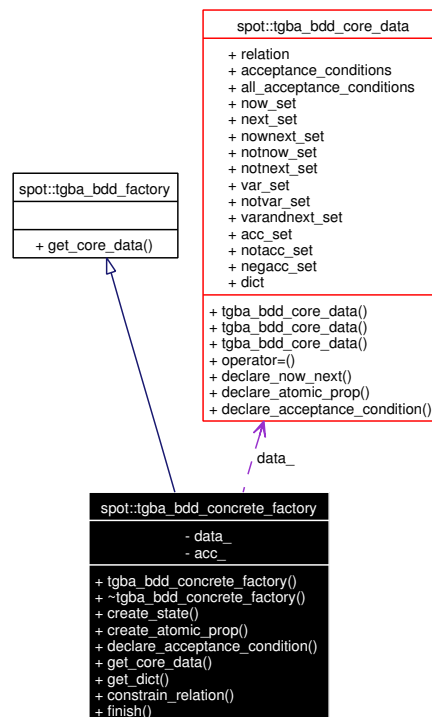
Helper class to build a `spot::tgba_bdd_concrete` object.

```
#include <tgba/tgabddconcretefactory.hh>
```

Inheritance diagram for `spot::tgba_bdd_concrete_factory`:



Collaboration diagram for `spot::tgba_bdd_concrete_factory`:



## Public Member Functions

- `tgba_bdd_concrete_factory (bdd_dict *dict)`
- `virtual ~tgba_bdd_concrete_factory ()`
- `int create_state (const ltl::formula *f)`
- `int create_atomic_prop (const ltl::formula *f)`
- `void declare_acceptance_condition (bdd b, const ltl::formula *a)`
- `const tgba_bdd_core_data & get_core_data () const`

*Get the core data for the new automata.*

- `bdd_dict * get_dict () const`
- `void constrain_relation (bdd new_rel)`  
*Add a new constraint to the relation.*
- `void finish ()`  
*Perform final computations before the relation can be used.*

### Private Types

- `typedef Sgi::hash_map< const ltl::formula *, bdd, ptr_hash< ltl::formula > > acc_map_`

### Private Attributes

- `tgba_bdd_core_data data_`  
*Core data for the new automata.*
- `acc_map_ acc_`  
*BDD associated to each acceptance condition.*

#### 11.91.1 Detailed Description

Helper class to build a `spot::tgba_bdd_concrete` object.

#### 11.91.2 Member Typedef Documentation

**11.91.2.1** `typedef Sgi::hash_map<const ltl::formula*, bdd, ptr_hash<ltl::formula> > spot::tgba_bdd_concrete_factory::acc_map_ [private]`

#### 11.91.3 Constructor & Destructor Documentation

**11.91.3.1** `spot::tgba_bdd_concrete_factory::tgba_bdd_concrete_factory (bdd_dict * dict)`

**11.91.3.2** `virtual spot::tgba_bdd_concrete_factory::~~tgba_bdd_concrete_factory () [virtual]`

#### 11.91.4 Member Function Documentation

**11.91.4.1** `void spot::tgba_bdd_concrete_factory::constrain_relation (bdd new_rel)`

Add a new constraint to the relation.

**11.91.4.2 int spot::tgba\_bdd\_concrete\_factory::create\_atomic\_prop (const [ltl::formula](#) \*f)**

Create an atomic proposition variable for formula  $f$ .

**Parameters:**

$f$  The formula to create an atomic proposition for.

**Returns:**

The variable number for this state.

The atomic proposition is not created if it already exists. Instead its existing variable number is returned. Variable numbers can be turned into BDD using `ithvar()`.

**11.91.4.3 int spot::tgba\_bdd\_concrete\_factory::create\_state (const [ltl::formula](#) \*f)**

Create a state variable for formula  $f$ .

**Parameters:**

$f$  The formula to create a state for.

**Returns:**

The variable number for this state.

The state is not created if it already exists. Instead its existing variable number is returned. Variable numbers can be turned into BDD using `ithvar()`.

**11.91.4.4 void spot::tgba\_bdd\_concrete\_factory::declare\_acceptance\_condition (bdd  $b$ , const [ltl::formula](#) \* $a$ )**

Declare an acceptance condition.

Formula such as ' $f \text{ U } g$ ' or ' $F g$ ' make the promise that ' $g$ ' will be fulfilled eventually. So once one of this formula has been translated into a BDD, we use `declare_acceptance_condition()` to associate all other states to the acceptance set of ' $g$ '.

**Parameters:**

$b$  a BDD indicating which variables are in the acceptance set

$a$  the formula associated

**11.91.4.5 void spot::tgba\_bdd\_concrete\_factory::finish ()**

Perform final computations before the relation can be used.

This function should be called after all propositions, state, acceptance conditions, and constraints have been declared, and before calling `get_code_data()` or `get_dict()`.

**11.91.4.6 const [tgba\\_bdd\\_core\\_data&](#) spot::tgba\_bdd\_concrete\_factory::get\_core\_data () const [virtual]**

Get the core data for the new automata.

Implements `spot::tgba_bdd_factory`.

11.91.4.7 [bdd\\_dict\\*](#) spot::tgba\_bdd\_concrete\_factory::get\_dict () const

### 11.91.5 Member Data Documentation

11.91.5.1 [acc\\_map\\_](#) spot::tgba\_bdd\_concrete\_factory::acc\_ [private]

BDD associated to each acceptance condition.

11.91.5.2 [tgba\\_bdd\\_core\\_data](#) spot::tgba\_bdd\_concrete\_factory::data\_ [private]

Core data for the new automata.

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

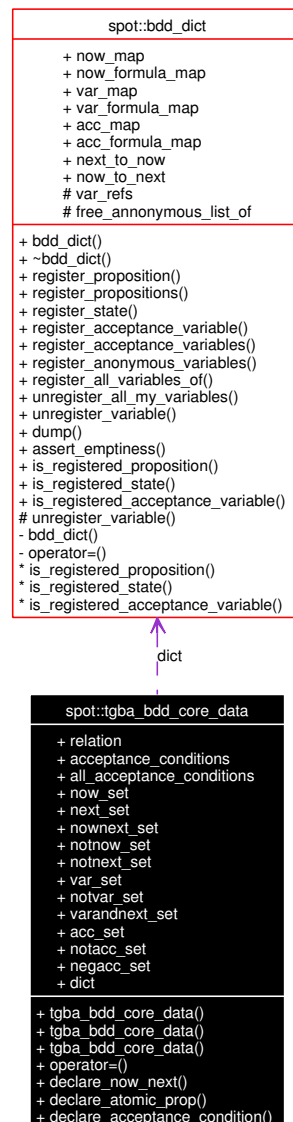
- [tgba/tgbabddconcretfactory.hh](#)

## 11.92 spot::tgba\_bdd\_core\_data Struct Reference

Core data for a TGBA encoded using BDDs.

```
#include <tgba/tgbabddcoredata.hh>
```

Collaboration diagram for spot::tgba\_bdd\_core\_data:



## Public Member Functions

- [tgba\\_bdd\\_core\\_data](#) ([bdd\\_dict](#) \*dict)  
*Default constructor.*
- [tgba\\_bdd\\_core\\_data](#) (const [tgba\\_bdd\\_core\\_data](#) &copy)  
*Copy constructor.*
- [tgba\\_bdd\\_core\\_data](#) (const [tgba\\_bdd\\_core\\_data](#) &left, const [tgba\\_bdd\\_core\\_data](#) &right)  
*Merge two [tgba\\_bdd\\_core\\_data](#).*
- const [tgba\\_bdd\\_core\\_data](#) & operator= (const [tgba\\_bdd\\_core\\_data](#) &copy)
- void [declare\\_now\\_next](#) (bdd now, bdd next)  
*Update the variable sets to take a new pair of variables into account.*

- void [declare\\_atomic\\_prop](#) (bdd var)  
*Update the variable sets to take a new atomic proposition into account.*
- void [declare\\_acceptance\\_condition](#) (bdd prom)  
*Update the variable sets to take a new acceptance condition into account.*

### Public Attributes

- bdd [relation](#)  
*encodes the transition relation of the TGBA.*
- bdd [acceptance\\_conditions](#)  
*encodes the acceptance conditions*
- bdd [all\\_acceptance\\_conditions](#)  
*The set of all acceptance conditions used by the Automaton.*
- bdd [now\\_set](#)  
*The conjunction of all Now variables, in their positive form.*
- bdd [next\\_set](#)  
*The conjunction of all Next variables, in their positive form.*
- bdd [nownext\\_set](#)  
*The conjunction of all Now and Next variables, in their positive form.*
- bdd [notnow\\_set](#)  
*The (positive) conjunction of all variables which are not Now variables.*
- bdd [notnext\\_set](#)  
*The (positive) conjunction of all variables which are not Next variables.*
- bdd [var\\_set](#)  
*The (positive) conjunction of all variables which are atomic propositions.*
- bdd [notvar\\_set](#)  
*The (positive) conjunction of all variables which are not atomic propositions.*
- bdd [varandnext\\_set](#)  
*The (positive) conjunction of all Next variables and atomic propositions.*
- bdd [acc\\_set](#)  
*The (positive) conjunction of all variables which are acceptance conditions.*
- bdd [notacc\\_set](#)  
*The (positive) conjunction of all variables which are not acceptance conditions.*
- bdd [negacc\\_set](#)



*The negative conjunction of all variables which are acceptance conditions.*

- `bdd_dict * dict`

*The dictionary used by the automata.*

### 11.92.1 Detailed Description

Core data for a TGBA encoded using BDDs.

### 11.92.2 Constructor & Destructor Documentation

#### 11.92.2.1 spot::tgba\_bdd\_core\_data::tgba\_bdd\_core\_data (`bdd_dict * dict`)

Default constructor.

Initially all variable set are empty and the `relation` is true.

#### 11.92.2.2 spot::tgba\_bdd\_core\_data::tgba\_bdd\_core\_data (const `tgba_bdd_core_data & copy`)

Copy constructor.

#### 11.92.2.3 spot::tgba\_bdd\_core\_data::tgba\_bdd\_core\_data (const `tgba_bdd_core_data & left`, const `tgba_bdd_core_data & right`)

Merge two `tgba_bdd_core_data`.

This is used when building a product of two automata.

### 11.92.3 Member Function Documentation

#### 11.92.3.1 void spot::tgba\_bdd\_core\_data::declare\_acceptance\_condition (`bdd prom`)

Update the variable sets to take a new acceptance condition into account.

#### 11.92.3.2 void spot::tgba\_bdd\_core\_data::declare\_atomic\_prop (`bdd var`)

Update the variable sets to take a new atomic proposition into account.

#### 11.92.3.3 void spot::tgba\_bdd\_core\_data::declare\_now\_next (`bdd now`, `bdd next`)

Update the variable sets to take a new pair of variables into account.

#### 11.92.3.4 const `tgba_bdd_core_data&` spot::tgba\_bdd\_core\_data::operator= (const `tgba_bdd_core_data & copy`)

### 11.92.4 Member Data Documentation

#### 11.92.4.1 `bdd spot::tgba_bdd_core_data::acc_set`

The (positive) conjunction of all variables which are acceptance conditions.

**11.92.4.2 bdd spot::tgba\_bdd\_core\_data::acceptance\_conditions**

encodes the acceptance conditions

$a \cup b$ , or  $\neg b$ , both imply that  $b$  should be verified eventually. We encode this with generalized Büchi accepting conditions. An acceptance set, called  $Acc[b]$ , hold all the state that do not promise to verify  $b$  eventually. (I.e., all the states that contain  $b$ , or do not contain  $a \cup b$ , or  $\neg b$ .)

The `spot::succ_iter::current_acceptance_conditions()` method will return the  $Acc[x]$  variables of the acceptance sets in which a transition is. Actually we never return  $Acc[x]$  alone, but  $Acc[x]$  and all other acceptance variables negated.

So if there is three acceptance set  $a$ ,  $b$ , and  $c$ , and a transition is in set  $a$ , we'll return  $Acc[a] \& !Acc[b] \& !Acc[c]$ . If the transition is in both  $a$  and  $b$ , we'll return  $(Acc[a] \& !Acc[b] \& !Acc[c]) \mid (!Acc[a] \& Acc[b] \& !Acc[c])$ .

Accepting conditions are attributed to transitions and are only concerned by atomic propositions (which label the transitions) and Next variables (the destination). Typically, a transition should bear the variable  $Acc[b]$  if it doesn't check for ' $b$ ' and have a destination of the form  $a \cup b$ , or  $\neg b$ .

To summarize, `acceptance_conditions` contains three kinds of variables:

- "Next" variables, that encode the destination state,
- atomic propositions, which are things to verify before going on to the next state,
- "Acc" variables.

**11.92.4.3 bdd spot::tgba\_bdd\_core\_data::all\_acceptance\_conditions**

The set of all acceptance conditions used by the Automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these accepting conditions. I.e., the union of the accepting conditions of all transition in the SCC should be equal to the result of this function.

**11.92.4.4 bdd dict\* spot::tgba\_bdd\_core\_data::dict**

The dictionary used by the automata.

**11.92.4.5 bdd spot::tgba\_bdd\_core\_data::negacc\_set**

The negative conjunction of all variables which are acceptance conditions.

**11.92.4.6 bdd spot::tgba\_bdd\_core\_data::next\_set**

The conjunction of all Next variables, in their positive form.

**11.92.4.7 bdd spot::tgba\_bdd\_core\_data::notacc\_set**

The (positive) conjunction of all variables which are not acceptance conditions.

**11.92.4.8 bdd spot::tgba\_bdd\_core\_data::notnext\_set**

The (positive) conjunction of all variables which are not Next variables.

**11.92.4.9** `bdd spot::tgba_bdd_core_data::notnow_set`

The (positive) conjunction of all variables which are not Now variables.

**11.92.4.10** `bdd spot::tgba_bdd_core_data::notvar_set`

The (positive) conjunction of all variables which are not atomic propositions.

**11.92.4.11** `bdd spot::tgba_bdd_core_data::now_set`

The conjunction of all Now variables, in their positive form.

**11.92.4.12** `bdd spot::tgba_bdd_core_data::nownext_set`

The conjunction of all Now and Next variables, in their positive form.

**11.92.4.13** `bdd spot::tgba_bdd_core_data::relation`

encodes the transition relation of the TGBA.

`relation` uses three kinds of variables:

- "Now" variables, that encode the current state
- "Next" variables, that encode the destination state
- atomic propositions, which are things to verify before going on to the next state

**11.92.4.14** `bdd spot::tgba_bdd_core_data::var_set`

The (positive) conjunction of all variables which are atomic propositions.

**11.92.4.15** `bdd spot::tgba_bdd_core_data::varandnext_set`

The (positive) conjunction of all Next variables and atomic propositions.

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

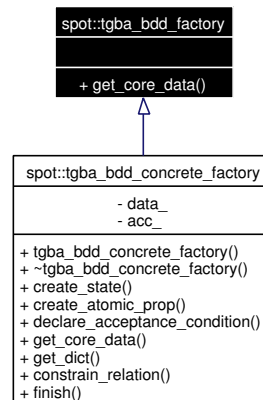
- [tgba/tgabddcoredata.hh](#)

**11.93** `spot::tgba_bdd_factory` Class Reference

Abstract class for `spot::tgba_bdd_concrete` factories.

```
#include <tgba/tgabddfactory.hh>
```

Inheritance diagram for `spot::tgba_bdd_factory`:



### Public Member Functions

- virtual const [tgba\\_bdd\\_core\\_data](#) & [get\\_core\\_data](#) () const =0

*Get the core data for the new automata.*

### 11.93.1 Detailed Description

Abstract class for [spot::tgba\\_bdd\\_concrete](#) factories.

A [spot::tgba\\_bdd\\_concrete](#) can be constructed from anything that supplies core data and their associated dictionary.

### 11.93.2 Member Function Documentation

**11.93.2.1** virtual const [tgba\\_bdd\\_core\\_data](#)& [spot::tgba\\_bdd\\_factory::get\\_core\\_data](#) () const [pure virtual]

Get the core data for the new automata.

Implemented in [spot::tgba\\_bdd\\_concrete\\_factory](#).

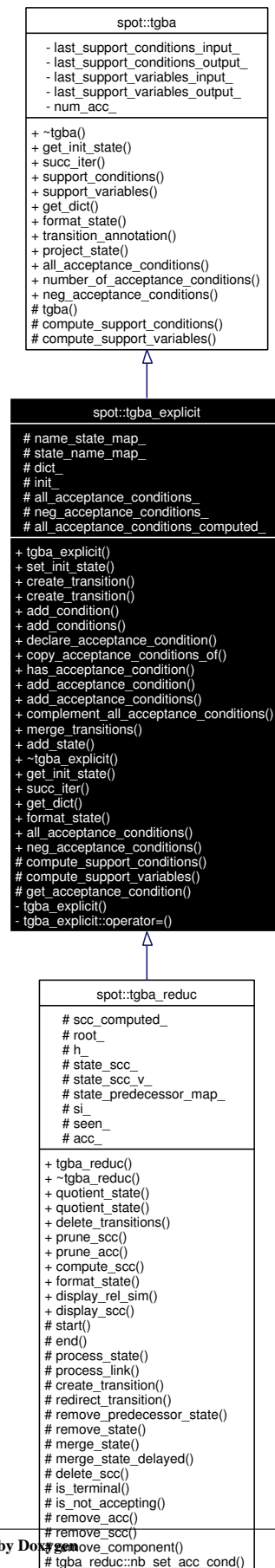
The documentation for this class was generated from the following file:

- [tgba/tgabddfactory.hh](#)

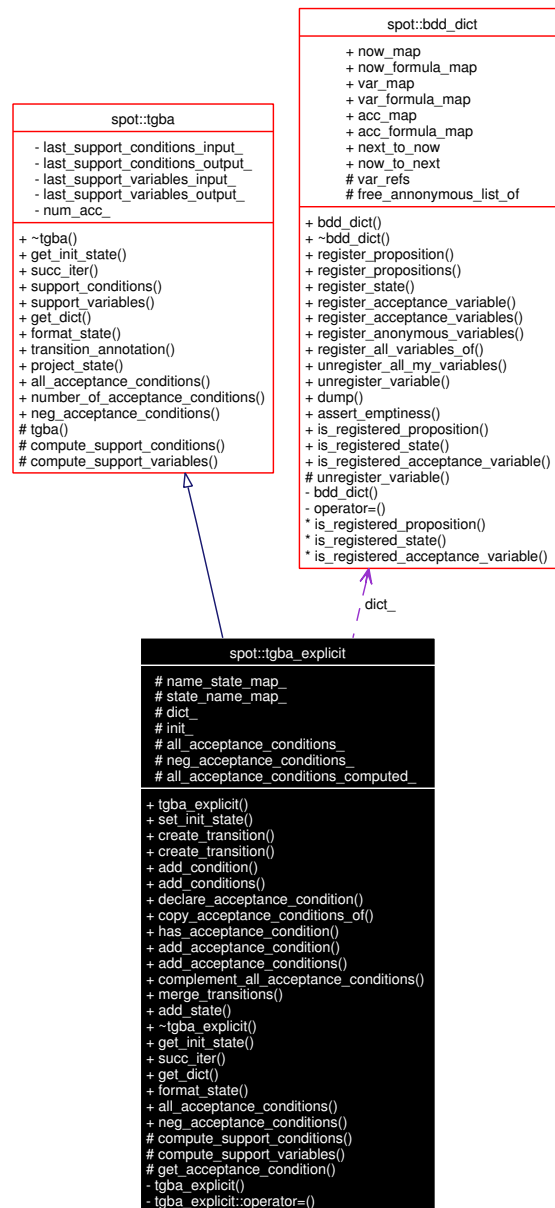
## 11.94 spot::tgba\_explicit Class Reference

```
#include <tgba/tgbaexplicit.hh>
```

Inheritance diagram for `spot::tgba_explicit`:



Collaboration diagram for spot::tgba\_explicit:



## Public Types

- typedef std::list< [transition](#) \* > [state](#)

## Public Member Functions

- [tgba\\_explicit](#) ([bdd\\_dict](#) \*dict)
- [state](#) \* [set\\_init\\_state](#) (const std::string &state)
- [transition](#) \* [create\\_transition](#) (const std::string &source, const std::string &dest)
- [transition](#) \* [create\\_transition](#) ([state](#) \*source, const [state](#) \*dest)

- void [add\\_condition](#) ([transition](#) \*t, const [ltl::formula](#) \*f)
- void [add\\_conditions](#) ([transition](#) \*t, bdd f)  
*This assumes that all variables in f are known from dict.*
- void [declare\\_acceptance\\_condition](#) (const [ltl::formula](#) \*f)
- void [copy\\_acceptance\\_conditions\\_of](#) (const [tgba](#) \*a)  
*Copy the acceptance conditions of a tgba.*
- bool [has\\_acceptance\\_condition](#) (const [ltl::formula](#) \*f) const
- void [add\\_acceptance\\_condition](#) ([transition](#) \*t, const [ltl::formula](#) \*f)
- void [add\\_acceptance\\_conditions](#) ([transition](#) \*t, bdd f)  
*This assumes that all acceptance conditions in f are known from dict.*
- void [complement\\_all\\_acceptance\\_conditions](#) ()
- void [merge\\_transitions](#) ()
- [state](#) \* [add\\_state](#) (const std::string &name)
- virtual [~tgba\\_explicit](#) ()
- virtual [spot::state](#) \* [get\\_init\\_state](#) () const  
*Get the initial state of the automaton.*
- virtual [tgba\\_succ\\_iterator](#) \* [succ\\_iter](#) (const [spot::state](#) \*local\_state, const [spot::state](#) \*global\_state=0, const [tgba](#) \*global\_automaton=0) const
- virtual bdd [dict](#) \* [get\\_dict](#) () const  
*Get the dictionary associated to the automaton.*
- virtual std::string [format\\_state](#) (const [spot::state](#) \*state) const
- virtual bdd [all\\_acceptance\\_conditions](#) () const  
*Return the set of all acceptance conditions used by this automaton.*
- virtual bdd [neg\\_acceptance\\_conditions](#) () const  
*Return the conjunction of all negated acceptance variables.*
- virtual [tgba\\_succ\\_iterator](#) \* [succ\\_iter](#) (const [state](#) \*local\_state, const [state](#) \*global\_state=0, const [tgba](#) \*global\_automaton=0) const =0  
*Get an iterator over the successors of local\_state.*
- bdd [support\\_conditions](#) (const [state](#) \*state) const  
*Get a formula that must hold whatever successor is taken.*
- bdd [support\\_variables](#) (const [state](#) \*state) const  
*Get the conjunctions of variables tested by the outgoing transitions of state.*
- virtual std::string [format\\_state](#) (const [state](#) \*state) const =0  
*Format the state as a string for printing.*
- virtual std::string [transition\\_annotation](#) (const [tgba\\_succ\\_iterator](#) \*t) const  
*Return a possible annotation for the transition pointed to by the iterator.*
- virtual [state](#) \* [project\\_state](#) (const [state](#) \*s, const [tgba](#) \*t) const  
*Project a state on an automaton.*

- virtual int [number\\_of\\_acceptance\\_conditions](#) () const

*The number of acceptance conditions.*

### Protected Types

- typedef Sgi::hash\_map< const std::string, [tgba\\_explicit::state](#) \*, [string\\_hash](#) > [ns\\_map](#)
- typedef Sgi::hash\_map< const [tgba\\_explicit::state](#) \*, std::string, [ptr\\_hash](#)< [tgba\\_explicit::state](#) > > [sn\\_map](#)

### Protected Member Functions

- virtual bdd [compute\\_support\\_conditions](#) (const [spot::state](#) \*state) const
- virtual bdd [compute\\_support\\_variables](#) (const [spot::state](#) \*state) const
- bdd [get\\_acceptance\\_condition](#) (const [ltl::formula](#) \*f)
- virtual bdd [compute\\_support\\_conditions](#) (const [state](#) \*state) const =0  
*Do the actual computation of [tgba::support\\_conditions\(\)](#).*
- virtual bdd [compute\\_support\\_variables](#) (const [state](#) \*state) const =0  
*Do the actual computation of [tgba::support\\_variables\(\)](#).*

### Protected Attributes

- [ns\\_map](#) [name\\_state\\_map\\_](#)
- [sn\\_map](#) [state\\_name\\_map\\_](#)
- bdd\_dict \* [dict\\_](#)
- [tgba\\_explicit::state](#) \* [init\\_](#)
- bdd [all\\_acceptance\\_conditions\\_](#)
- bdd [neg\\_acceptance\\_conditions\\_](#)
- bool [all\\_acceptance\\_conditions\\_computed\\_](#)

### Private Member Functions

- [tgba\\_explicit](#) (const [tgba\\_explicit](#) &other)
- [tgba\\_explicit](#) & [tgba\\_explicit::operator=](#) (const [tgba\\_explicit](#) &other)

### Classes

- struct [transition](#)  
*Explicit transitions (used by [spot::tgba\\_explicit](#)).*

#### 11.94.1 Detailed Description

Explicit representation of a [spot::tgba](#).



### 11.94.2 Member Typedef Documentation

**11.94.2.1** `typedef Sgi::hash_map<const std::string, tgba_explicit::state*, string_hash> spot::tgba_explicit::ns_map` [protected]

**11.94.2.2** `typedef Sgi::hash_map<const tgba_explicit::state*, std::string, ptr_hash<tgba_explicit::state>> spot::tgba_explicit::sn_map` [protected]

**11.94.2.3** `typedef std::list<transition*> spot::tgba_explicit::state`

### 11.94.3 Constructor & Destructor Documentation

**11.94.3.1** `spot::tgba_explicit::tgba_explicit (bdd_dict * dict)`

**11.94.3.2** `virtual spot::tgba_explicit::~~tgba_explicit ()` [virtual]

**11.94.3.3** `spot::tgba_explicit::tgba_explicit (const tgba_explicit & other)` [private]

### 11.94.4 Member Function Documentation

**11.94.4.1** `void spot::tgba_explicit::add_acceptance_condition (transition * t, const ltl::formula * f)`

**11.94.4.2** `void spot::tgba_explicit::add_acceptance_conditions (transition * t, bdd f)`

This assumes that all acceptance conditions in  $f$  are known from dict.

**11.94.4.3** `void spot::tgba_explicit::add_condition (transition * t, const ltl::formula * f)`

**11.94.4.4** `void spot::tgba_explicit::add_conditions (transition * t, bdd f)`

This assumes that all variables in  $f$  are known from dict.

**11.94.4.5** `state* spot::tgba_explicit::add_state (const std::string & name)`

Return the `tgba_explicit::state` for  $name$ , creating the state if it does not exist.

**11.94.4.6** `virtual bdd spot::tgba_explicit::all_acceptance_conditions () const` [virtual]

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implements `spot::tgba`.

**11.94.4.7** `void spot::tgba_explicit::complement_all_acceptance_conditions ()`

**11.94.4.8** virtual bdd spot::tgba::compute\_support\_conditions (const [state](#) \* *state*) const  
[protected, pure virtual, inherited]

Do the actual computation of [tgba::support\\_conditions\(\)](#).

Implemented in [spot::tgba\\_bdd\\_concrete](#), [spot::tgba\\_product](#), and [spot::tgba\\_tba\\_proxy](#).

**11.94.4.9** virtual bdd spot::tgba\_explicit::compute\_support\_conditions (const [spot::state](#) \* *state*) const  
[protected, virtual]

**11.94.4.10** virtual bdd spot::tgba::compute\_support\_variables (const [state](#) \* *state*) const  
[protected, pure virtual, inherited]

Do the actual computation of [tgba::support\\_variables\(\)](#).

Implemented in [spot::tgba\\_bdd\\_concrete](#), [spot::tgba\\_product](#), and [spot::tgba\\_tba\\_proxy](#).

**11.94.4.11** virtual bdd spot::tgba\_explicit::compute\_support\_variables (const [spot::state](#) \* *state*) const  
[protected, virtual]

**11.94.4.12** void spot::tgba\_explicit::copy\_acceptance\_conditions\_of (const [tgba](#) \* *a*)

Copy the acceptance conditions of a tgba.

If used, this function should be called before creating any transition.

**11.94.4.13** [transition\\*](#) spot::tgba\_explicit::create\_transition ([state](#) \* *source*, const [state](#) \* *dest*)

**11.94.4.14** [transition\\*](#) spot::tgba\_explicit::create\_transition (const std::string & *source*, const std::string & *dest*)

**11.94.4.15** void spot::tgba\_explicit::declare\_acceptance\_condition (const [ltl::formula](#) \* *f*)

**11.94.4.16** virtual std::string spot::tgba::format\_state (const [state](#) \* *state*) const [pure virtual, inherited]

Format the state as a string for printing.

This formatting is the responsibility of the automata who owns the state.

Implemented in [spot::tgba\\_bdd\\_concrete](#), [spot::tgba\\_product](#), and [spot::tgba\\_tba\\_proxy](#).

**11.94.4.17** virtual std::string spot::tgba\_explicit::format\_state (const [spot::state](#) \* *state*) const  
[virtual]

Reimplemented in [spot::tgba\\_reduc](#).

**11.94.4.18** bdd spot::tgba\_explicit::get\_acceptance\_condition (const [ltl::formula](#) \* *f*)  
[protected]

**11.94.4.19 virtual [bdd\\_dict\\*](#) spot::tgba\_explicit::get\_dict () const [virtual]**

Get the dictionary associated to the automaton.

State are represented as BDDs. The dictionary allows to map BDD variables back to formulae, and vice versa. This is useful when dealing with several automata (which may use the same BDD variable for different formula), or simply when printing.

Implements [spot::tgba](#).

**11.94.4.20 virtual [spot::state\\*](#) spot::tgba\_explicit::get\_init\_state () const [virtual]**

Get the initial state of the automaton.

The state has been allocated with new. It is the responsibility of the caller to delete it when no longer needed.

Implements [spot::tgba](#).

**11.94.4.21 bool spot::tgba\_explicit::has\_acceptance\_condition (const [ltl::formula](#) \* f) const****11.94.4.22 void spot::tgba\_explicit::merge\_transitions ()****11.94.4.23 virtual [bdd](#) spot::tgba\_explicit::neg\_acceptance\_conditions () const [virtual]**

Return the conjunction of all negated acceptance variables.

For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a]&!Acc[b]&!Acc[c]`.

This is useful when making products: each operand's condition set should be augmented with the [neg\\_acceptance\\_conditions\(\)](#) of the other operand.

Implements [spot::tgba](#).

**11.94.4.24 virtual int spot::tgba::number\_of\_acceptance\_conditions () const [virtual, inherited]**

The number of acceptance conditions.

**11.94.4.25 virtual [state\\*](#) spot::tgba::project\_state (const [state](#) \* s, const [tgba](#) \* t) const [virtual, inherited]**

Project a state on an automaton.

This converts *s*, into that corresponding [spot::state](#) for *t*. This is useful when you have the state of a product, and want restrict this state to a specific automata occurring in the product.

It goes without saying that *s* and *t* should be compatible (i.e., *s* is a state of *t*).

**Returns:**

0 if the projection fails (*s* is unrelated to *t*), or a new `state*` (the projected state) that must be deleted by the caller.

Reimplemented in [spot::tgba\\_product](#), and [spot::tgba\\_tba\\_proxy](#).

**11.94.4.26** `state*` `spot::tgba_explicit::set_init_state (const std::string & state)`

**11.94.4.27** `virtual tgba_succ_iterator*` `spot::tgba::succ_iter (const state * local_state, const state * global_state = 0, const tgba * global_automaton = 0) const` [pure virtual, inherited]

Get an iterator over the successors of *local\_state*.

The iterator has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

During synchornized products, additional informations are passed about the entire product and its state. Recall that products can be nested, forming a tree of `spot::tgba` where most values are computed on demand. *global\_automaton* designate the root `spot::tgba`, and *global\_state* its state. This two objects can be used by `succ_iter()` to restrict the set of successors to compute.

**Parameters:**

*local\_state* The state whose successors are to be explored. This pointer is not adopted in any way by `succ_iter`, and it is still the caller's responsibility to delete it when appropriate (this can be done during the lifetime of the iterator).

*global\_state* In a product, the state of the global product automaton. Otherwise, 0. Like *locale\_state*, *global\_state* is not adopted by `succ_iter`.

*global\_automaton* In a product, the global product automaton. Otherwise, 0.

Implemented in `spot::tgba_bdd_concrete`, `spot::tgba_product`, and `spot::tgba_tba_proxy`.

**11.94.4.28** `virtual tgba_succ_iterator*` `spot::tgba_explicit::succ_iter (const spot::state * local_state, const spot::state * global_state = 0, const tgba * global_automaton = 0) const` [virtual]

**11.94.4.29** `bdd` `spot::tgba::support_conditions (const state * state) const` [inherited]

Get a formula that must hold whatever successor is taken.

**Returns:**

A formula which must be verified for all successors of *state*.

This can be as simple as `bddtrue`, or more completely the disjunction of the condition of all successors. This is used as an hint by `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_conditions()`, this function is just a wrapper that will cache the last return value for efficiency.

**11.94.4.30** `bdd` `spot::tgba::support_variables (const state * state) const` [inherited]

Get the conjunctions of variables tested by the outgoing transitions of *state*.

All variables tested by outgoing transitions must be returned. This is mandatory.

This is used as an hint by some `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_variables()`, this function is just a wrapper that will cache the last return value for efficiency.

**11.94.4.31** `tgba_explicit&` `spot::tgba_explicit::tgba_explicit::operator= (const tgba_explicit & other)` [private]

**11.94.4.32** `virtual std::string spot::tgba::transition_annotation (const tgba\_succ\_iterator * t) const` `[virtual, inherited]`

Return a possible annotation for the transition pointed to by the iterator.

Implementing this function is optional; the default annotation is the empty string.

**Parameters:**

*t* a non-done [tgba\\_succ\\_iterator](#) for this automata

Reimplemented in [spot::tgba\\_product](#), and [spot::tgba\\_tba\\_proxy](#).

## 11.94.5 Member Data Documentation

**11.94.5.1** `bdd spot::tgba\_explicit::all\_acceptance\_conditions\_` `[mutable, protected]`

**11.94.5.2** `bool spot::tgba\_explicit::all\_acceptance\_conditions\_computed\_` `[mutable, protected]`

**11.94.5.3** `bdd_dict* spot::tgba\_explicit::dict\_` `[protected]`

**11.94.5.4** `tgba\_explicit::state\* spot::tgba\_explicit::init\_` `[protected]`

**11.94.5.5** `ns_map spot::tgba\_explicit::name\_state\_map\_` `[protected]`

**11.94.5.6** `bdd spot::tgba\_explicit::neg\_acceptance\_conditions\_` `[protected]`

**11.94.5.7** `sn_map spot::tgba\_explicit::state\_name\_map\_` `[protected]`

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

- [tgba/tgbaexplicit.hh](#)

## 11.95 `spot::tgba_explicit::transition` Struct Reference

Explicit transitions (used by [spot::tgba\\_explicit](#)).

```
#include <tgba/tgbaexplicit.hh>
```

### Public Attributes

- bdd [condition](#)
- bdd [acceptance\\_conditions](#)
- const [state](#) \* [dest](#)

### 11.95.1 Detailed Description

Explicit transitions (used by [spot::tgba\\_explicit](#)).

## 11.95.2 Member Data Documentation

### 11.95.2.1 bdd [spot::tgba\\_explicit::transition::acceptance\\_conditions](#)

### 11.95.2.2 bdd [spot::tgba\\_explicit::transition::condition](#)

### 11.95.2.3 const [state\\*](#) [spot::tgba\\_explicit::transition::dest](#)

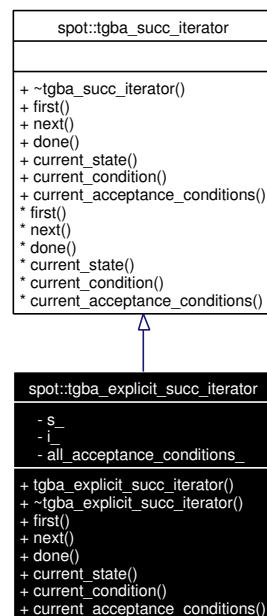
The documentation for this struct was generated from the following file:

- [tgba/tgbaexplicit.hh](#)

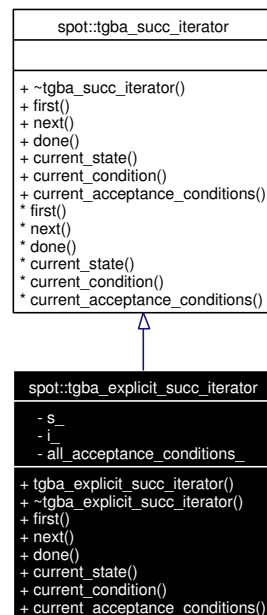
## 11.96 spot::tgba\_explicit\_succ\_iterator Class Reference

```
#include <tgba/tgbaexplicit.hh>
```

Inheritance diagram for `spot::tgba_explicit_succ_iterator`:



Collaboration diagram for `spot::tgba_explicit_succ_iterator`:



## Public Member Functions

- [tgba\\_explicit\\_succ\\_iterator](#) (const [tgba\\_explicit::state](#) \*s, bdd all\_acc)
- virtual [~tgba\\_explicit\\_succ\\_iterator](#) ()
- virtual void [first](#) ()  
*Position the iterator on the first successor (if any).*
- virtual void [next](#) ()  
*Jump to the next successor (if any).*
- virtual bool [done](#) () const  
*Check whether the iteration is finished.*
- virtual [state\\_explicit](#) \* [current\\_state](#) () const  
*Get the state of the current successor.*
- virtual bdd [current\\_condition](#) () const  
*Get the condition on the transition leading to this successor.*
- virtual bdd [current\\_acceptance\\_conditions](#) () const  
*Get the acceptance conditions on the transition leading to this successor.*

## Private Attributes

- const [tgba\\_explicit::state](#) \* [s\\_](#)
- [tgba\\_explicit::state::const\\_iterator](#) [i\\_](#)
- bdd [all\\_acceptance\\_conditions\\_](#)

### 11.96.1 Detailed Description

Successor iterators used by [spot::tgba\\_explicit](#).

### 11.96.2 Constructor & Destructor Documentation

**11.96.2.1** `spot::tgba_explicit_succ_iterator::tgba_explicit_succ_iterator (const tgba\_explicit::state * s, bdd all\_acc)`

**11.96.2.2** `virtual spot::tgba_explicit_succ_iterator::~tgba\_explicit\_succ\_iterator ()` `[inline, virtual]`

### 11.96.3 Member Function Documentation

**11.96.3.1** `virtual bdd spot::tgba_explicit_succ_iterator::current_acceptance_conditions ()` `const` `[virtual]`

Get the acceptance conditions on the transition leading to this successor.

Implements [spot::tgba\\_succ\\_iterator](#).

**11.96.3.2** `virtual bdd spot::tgba_explicit_succ_iterator::current_condition ()` `const` `[virtual]`

Get the condition on the transition leading to this successor.

This is a boolean function of atomic propositions.

Implements [spot::tgba\\_succ\\_iterator](#).

**11.96.3.3** `virtual state\_explicit\* spot::tgba_explicit_succ_iterator::current_state ()` `const` `[virtual]`

Get the state of the current successor.

Note that the same state may occur at different points in the iteration. These actually correspond to the same destination. It just means there were several transitions, with different conditions, leading to the same state.

Implements [spot::tgba\\_succ\\_iterator](#).

**11.96.3.4** `virtual bool spot::tgba_explicit_succ_iterator::done ()` `const` `[virtual]`

Check whether the iteration is finished.

This function should be called after any call to [first\(\)](#) or [next\(\)](#) and before any enquiry about the current state.

The usual way to do this is with a `for` loop.

```
for (s->first(); !s->done(); s->next())
    ...
```

Implements [spot::tgba\\_succ\\_iterator](#).



**11.96.3.5** `virtual void spot::tgba_explicit_succ_iterator::first ()` [virtual]

Position the iterator on the first successor (if any).

This method can be called several times to make multiple passes over successors.

**Warning:**

One should always call `done()` to ensure there is a successor, even after `first()`. A common trap is to assume that there is at least one successor: this is wrong.

Implements `spot::tgba_succ_iterator`.

**11.96.3.6** `virtual void spot::tgba_explicit_succ_iterator::next ()` [virtual]

Jump to the next successor (if any).

**Warning:**

Again, one should always call `done()` to ensure there is a successor.

Implements `spot::tgba_succ_iterator`.

**11.96.4** Member Data Documentation**11.96.4.1** `bdd spot::tgba_explicit_succ_iterator::all_acceptance_conditions_` [private]**11.96.4.2** `tgba_explicit::state::const_iterator spot::tgba_explicit_succ_iterator::i_` [private]**11.96.4.3** `const tgba_explicit::state* spot::tgba_explicit_succ_iterator::s_` [private]

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

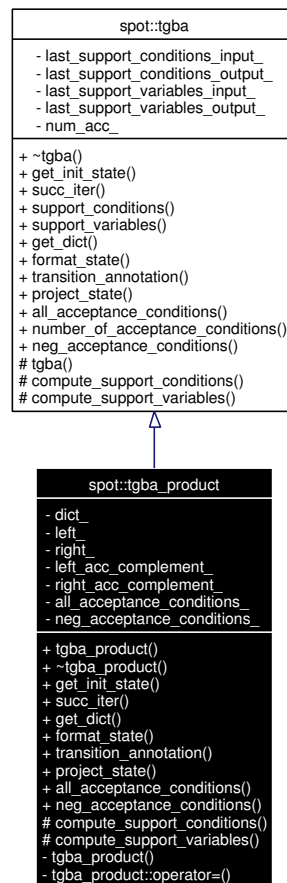
- `tgba/tgbaexplicit.hh`

**11.97** `spot::tgba_product` Class Reference

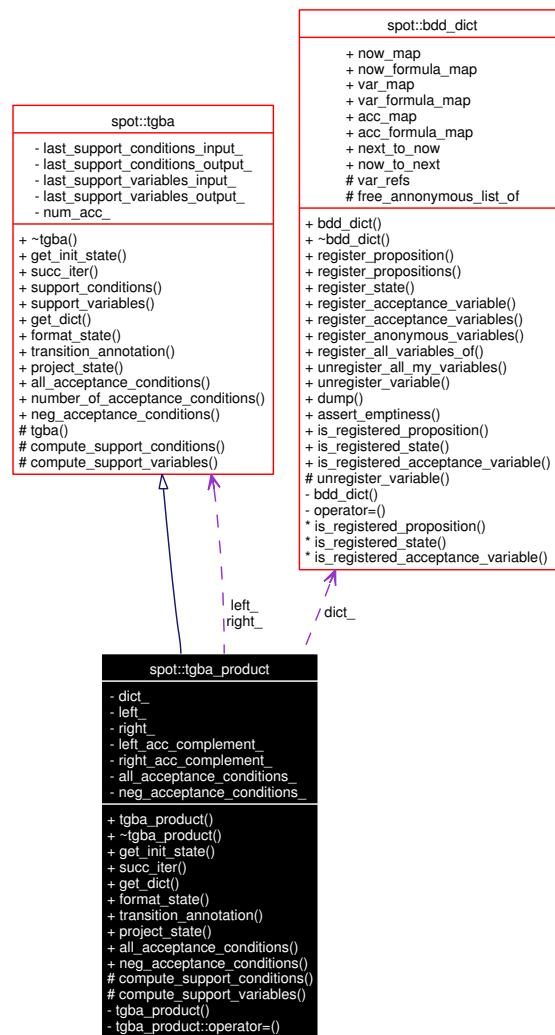
A lazy product. (States are computed on the fly.).

```
#include <tgba/tgbaproduct.hh>
```

Inheritance diagram for `spot::tgba_product`:



Collaboration diagram for spot::tgba\_product:



## Public Member Functions

- **tgba\_product** (const **tgba** \*left, const **tgba** \*right)  
*Constructor.*
- virtual **~tgba\_product** ()
- virtual **state** \* **get\_init\_state** () const  
*Get the initial state of the automaton.*
- virtual **tgba\_succ\_iterator\_product** \* **succ\_iter** (const **state** \*local\_state, const **state** \*global\_state=0, const **tgba** \*global\_automaton=0) const  
*Get an iterator over the successors of local\_state.*
- virtual **bdd\_dict** \* **get\_dict** () const  
*Get the dictionary associated to the automaton.*
- virtual std::string **format\_state** (const **state** \*state) const

*Format the state as a string for printing.*

- virtual std::string [transition\\_annotation](#) (const [tgba\\_succ\\_iterator](#) \*t) const  
*Return a possible annotation for the transition pointed to by the iterator.*
- virtual [state](#) \* [project\\_state](#) (const [state](#) \*s, const [tgba](#) \*t) const  
*Project a state on an automaton.*
- virtual bdd [all\\_acceptance\\_conditions](#) () const  
*Return the set of all acceptance conditions used by this automaton.*
- virtual bdd [neg\\_acceptance\\_conditions](#) () const  
*Return the conjunction of all negated acceptance variables.*
- bdd [support\\_conditions](#) (const [state](#) \*state) const  
*Get a formula that must hold whatever successor is taken.*
- bdd [support\\_variables](#) (const [state](#) \*state) const  
*Get the conjunctions of variables tested by the outgoing transitions of state.*
- virtual int [number\\_of\\_acceptance\\_conditions](#) () const  
*The number of acceptance conditions.*

### Protected Member Functions

- virtual bdd [compute\\_support\\_conditions](#) (const [state](#) \*state) const  
*Do the actual computation of [tgba::support\\_conditions\(\)](#).*
- virtual bdd [compute\\_support\\_variables](#) (const [state](#) \*state) const  
*Do the actual computation of [tgba::support\\_variables\(\)](#).*

### Private Member Functions

- [tgba\\_product](#) (const [tgba\\_product](#) &)
- [tgba\\_product](#) & [tgba\\_product::operator=](#) (const [tgba\\_product](#) &)

### Private Attributes

- [bdd\\_dict](#) \* [dict\\_](#)
- const [tgba](#) \* [left\\_](#)
- const [tgba](#) \* [right\\_](#)
- bdd [left\\_acc\\_complement\\_](#)
- bdd [right\\_acc\\_complement\\_](#)
- bdd [all\\_acceptance\\_conditions\\_](#)
- bdd [neg\\_acceptance\\_conditions\\_](#)

### 11.97.1 Detailed Description

A lazy product. (States are computed on the fly.).

### 11.97.2 Constructor & Destructor Documentation

#### 11.97.2.1 spot::tgba\_product::tgba\_product (const tgba \* left, const tgba \* right)

Constructor.

##### Parameters:

*left* The left automata in the product.

*right* The right automata in the product. Do not be fooled by these arguments: a product is commutative.

#### 11.97.2.2 virtual spot::tgba\_product::~~tgba\_product () [virtual]

#### 11.97.2.3 spot::tgba\_product::tgba\_product (const tgba\_product &) [private]

### 11.97.3 Member Function Documentation

#### 11.97.3.1 virtual bdd spot::tgba\_product::all\_acceptance\_conditions () const [virtual]

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::tgba](#).

#### 11.97.3.2 virtual bdd spot::tgba\_product::compute\_support\_conditions (const state \* state) const [protected, virtual]

Do the actual computation of [tgba::support\\_conditions\(\)](#).

Implements [spot::tgba](#).

#### 11.97.3.3 virtual bdd spot::tgba\_product::compute\_support\_variables (const state \* state) const [protected, virtual]

Do the actual computation of [tgba::support\\_variables\(\)](#).

Implements [spot::tgba](#).

#### 11.97.3.4 virtual std::string spot::tgba\_product::format\_state (const state \* state) const [virtual]

Format the state as a string for printing.

This formatting is the responsibility of the automata who owns the state.

Implements [spot::tgba](#).

**11.97.3.5 virtual [bdd\\_dict\\*](#) spot::tgba\_product::get\_dict () const [virtual]**

Get the dictionary associated to the automaton.

State are represented as BDDs. The dictionary allows to map BDD variables back to formulae, and vice versa. This is useful when dealing with several automata (which may use the same BDD variable for different formula), or simply when printing.

Implements [spot::tgba](#).

**11.97.3.6 virtual [state\\*](#) spot::tgba\_product::get\_init\_state () const [virtual]**

Get the initial state of the automaton.

The state has been allocated with new. It is the responsibility of the caller to delete it when no longer needed.

Implements [spot::tgba](#).

**11.97.3.7 virtual [bdd](#) spot::tgba\_product::neg\_acceptance\_conditions () const [virtual]**

Return the conjunction of all negated acceptance variables.

For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a]&!Acc[b]&!Acc[c]`.

This is useful when making products: each operand's condition set should be augmented with the [neg\\_acceptance\\_conditions\(\)](#) of the other operand.

Implements [spot::tgba](#).

**11.97.3.8 virtual int spot::tgba::number\_of\_acceptance\_conditions () const [virtual, inherited]**

The number of acceptance conditions.

**11.97.3.9 virtual [state\\*](#) spot::tgba\_product::project\_state (const [state](#) \* s, const [tgba](#) \* t) const [virtual]**

Project a state on an automaton.

This converts *s*, into that corresponding [spot::state](#) for *t*. This is useful when you have the state of a product, and want restrict this state to a specific automata occurring in the product.

It goes without saying that *s* and *t* should be compatible (i.e., *s* is a state of *t*).

**Returns:**

0 if the projection fails (*s* is unrelated to *t*), or a new `state*` (the projected state) that must be deleted by the caller.

Reimplemented from [spot::tgba](#).

**11.97.3.10 virtual [tgba\\_succ\\_iterator\\_product\\*](#) spot::tgba\_product::succ\_iter (const [state](#) \* local\_state, const [state](#) \* global\_state = 0, const [tgba](#) \* global\_automaton = 0) const [virtual]**

Get an iterator over the successors of *local\_state*.

The iterator has been allocated with new. It is the responsibility of the caller to delete it when no longer needed.

During synchornized products, additional informations are passed about the entire product and its state. Recall that products can be nested, forming a tree of `spot::tgba` where most values are computed on demand. `global_automaton` designate the root `spot::tgba`, and `global_state` its state. This two objects can be used by `succ_iter()` to restrict the set of successors to compute.

**Parameters:**

*local\_state* The state whose successors are to be explored. This pointer is not adopted in any way by `succ_iter`, and it is still the caller's responsibility to delete it when appropriate (this can be done during the lifetime of the iterator).

*global\_state* In a product, the state of the global product automaton. Otherwise, 0. Like *locale\_state*, `global_state` is not adopted by `succ_iter`.

*global\_automaton* In a product, the global product automaton. Otherwise, 0.

Implements `spot::tgba`.

### 11.97.3.11 `bdd spot::tgba::support_conditions (const state * state) const` [inherited]

Get a formula that must hold whatever successor is taken.

**Returns:**

A formula which must be verified for all successors of *state*.

This can be as simple as `bddtrue`, or more completely the disjunction of the condition of all successors. This is used as an hint by `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_conditions()`, this function is just a wrapper that will cache the last return value for efficiency.

### 11.97.3.12 `bdd spot::tgba::support_variables (const state * state) const` [inherited]

Get the conjunctions of variables tested by the outgoing transitions of *state*.

All variables tested by outgoing transitions must be returned. This is mandatory.

This is used as an hint by some `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_variables()`, this function is just a wrapper that will cache the last return value for efficiency.

### 11.97.3.13 `tgba_product& spot::tgba_product::tgba_product::operator= (const tgba_product &) [private]`

### 11.97.3.14 `virtual std::string spot::tgba_product::transition_annotation (const tgba_succ_iterator * t) const` [virtual]

Return a possible annotation for the transition pointed to by the iterator.

Implementing this function is optional; the default annotation it the empty string.

**Parameters:**

*t* a non-done `tgba_succ_iterator` for this automata

Reimplemented from `spot::tgba`.

### 11.97.4 Member Data Documentation

11.97.4.1 bdd [spot::tgba\\_product::all\\_acceptance\\_conditions\\_](#) [private]

11.97.4.2 bdd\_dict\* [spot::tgba\\_product::dict\\_](#) [private]

11.97.4.3 const [tgba\\*](#) [spot::tgba\\_product::left\\_](#) [private]

11.97.4.4 bdd [spot::tgba\\_product::left\\_acc\\_complement\\_](#) [private]

11.97.4.5 bdd [spot::tgba\\_product::neg\\_acceptance\\_conditions\\_](#) [private]

11.97.4.6 const [tgba\\*](#) [spot::tgba\\_product::right\\_](#) [private]

11.97.4.7 bdd [spot::tgba\\_product::right\\_acc\\_complement\\_](#) [private]

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

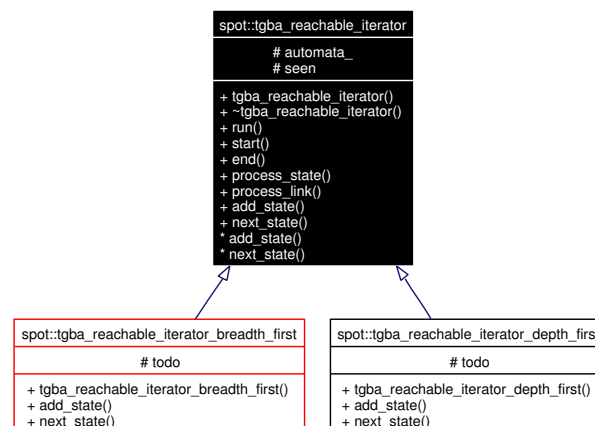
- [tgba/tgbaproduct.hh](#)

## 11.98 spot::tgba\_reachable\_iterator Class Reference

Iterate over all reachable states of a [spot::tgba](#).

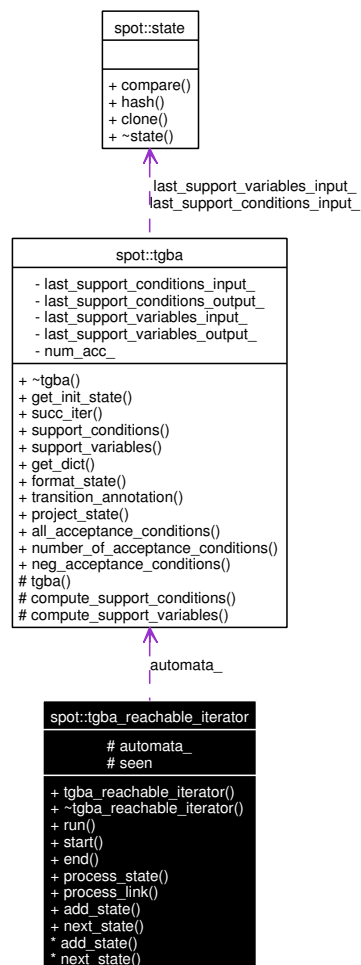
```
#include <tgbaalgos/reachiter.hh>
```

Inheritance diagram for [spot::tgba\\_reachable\\_iterator](#):



Collaboration diagram for [spot::tgba\\_reachable\\_iterator](#):





## Public Member Functions

- `tgba_reachable_iterator` (const `tgba` \*a)
- virtual `~tgba_reachable_iterator` ()
- void `run` ()  
*Iterate over all reachable states of a `spot::tgba`.*
- virtual void `start` ()  
*Called by `run()` before starting its iteration.*
- virtual void `end` ()  
*Called by `run()` once all states have been explored.*
- virtual void `process_state` (const `state` \*s, int n, `tgba_succ_iterator` \*si)
- virtual void `process_link` (const `state` \*in\_s, int in, const `state` \*out\_s, int out, const `tgba_succ_iterator` \*si)

## Todo list management.

*Called by `run()` to register newly discovered states.*

*spot::tgba\_reachable\_iterator\_depth\_first* and *spot::tgba\_reachable\_iterator\_breadth\_first* offer two precanned implementations for these functions.

- virtual void `add_state` (const `state` \*s)=0
- virtual const `state` \* `next_state` ()=0

*Called by `run()` to obtain the.*

### Protected Types

- typedef Sgi::hash\_map< const `state` \*, int, `state_ptr_hash`, `state_ptr_equal` > `seen_map`

### Protected Attributes

- const `tgba` \* `automata_`  
*The `spot::tgba` to explore.*
- `seen_map` `seen`  
*States already seen.*

## 11.98.1 Detailed Description

Iterate over all reachable states of a `spot::tgba`.

## 11.98.2 Member Typedef Documentation

**11.98.2.1** typedef Sgi::hash\_map<const `state`\*, int, `state_ptr_hash`, `state_ptr_equal`> `spot::tgba_reachable_iterator::seen_map` [protected]

Reimplemented in `spot::tgba_reduc`.

## 11.98.3 Constructor & Destructor Documentation

**11.98.3.1** `spot::tgba_reachable_iterator::tgba_reachable_iterator` (const `tgba` \* *a*)

**11.98.3.2** virtual `spot::tgba_reachable_iterator::~~tgba_reachable_iterator` () [virtual]

## 11.98.4 Member Function Documentation

**11.98.4.1** virtual void `spot::tgba_reachable_iterator::add_state` (const `state` \* *s*) [pure virtual]

Implemented in `spot::tgba_reachable_iterator_depth_first`, and `spot::tgba_reachable_iterator_breadth_first`.

**11.98.4.2** virtual void `spot::tgba_reachable_iterator::end` () [virtual]

Called by `run()` once all states have been explored.

Reimplemented in `spot::tgba_reduc`, and `spot::parity_game_graph`.

**11.98.4.3 virtual const [state](#)\* spot::tgba\_reachable\_iterator::next\_state ()** [pure virtual]

Called by [run\(\)](#) to obtain the.

Implemented in [spot::tgba\\_reachable\\_iterator\\_depth\\_first](#), and [spot::tgba\\_reachable\\_iterator\\_breadth\\_first](#).

**11.98.4.4 virtual void spot::tgba\_reachable\_iterator::process\_link (const [state](#) \* *in\_s*, int *in*, const [state](#) \* *out\_s*, int *out*, const [tgba\\_succ\\_iterator](#) \* *si*)** [virtual]

Called by [run\(\)](#) to process a transition.

**Parameters:**

*in\_s* The source state

*in* The source state number.

*out\_s* The destination state

*out* The destination state number.

*si* The [spot::tgba\\_succ\\_iterator](#) positionned on the current transition.

The *in\_s* and *out\_s* states are owned by the [spot::tgba\\_reachable\\_iterator](#) instance and destroyed when the instance is destroyed.

**11.98.4.5 virtual void spot::tgba\_reachable\_iterator::process\_state (const [state](#) \* *s*, int *n*, [tgba\\_succ\\_iterator](#) \* *si*)** [virtual]

Called by [run\(\)](#) to process a state.

**Parameters:**

*s* The current state.

*n* A unique number assigned to *s*.

*si* The [spot::tgba\\_succ\\_iterator](#) for *s*.

Reimplemented in [spot::parity\\_game\\_graph](#).

**11.98.4.6 void spot::tgba\_reachable\_iterator::run ()**

Iterate over all reachable states of a [spot::tgba](#).

This is a template method that will call [add\\_state\(\)](#), [next\\_state\(\)](#), [start\(\)](#), [end\(\)](#), [process\\_state\(\)](#), and [process\\_link\(\)](#), while it iterate over state.

**11.98.4.7 virtual void spot::tgba\_reachable\_iterator::start ()** [virtual]

Called by [run\(\)](#) before starting its iteration.

Reimplemented in [spot::tgba\\_reduc](#), and [spot::parity\\_game\\_graph](#).

**11.98.5 Member Data Documentation****11.98.5.1 const [tgba](#)\* spot::tgba\_reachable\_iterator::automata\_** [protected]

The [spot::tgba](#) to explore.

## 11.98.5.2 seen\_map spot::tgba\_reachable\_iterator::seen [protected]

States already seen.

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

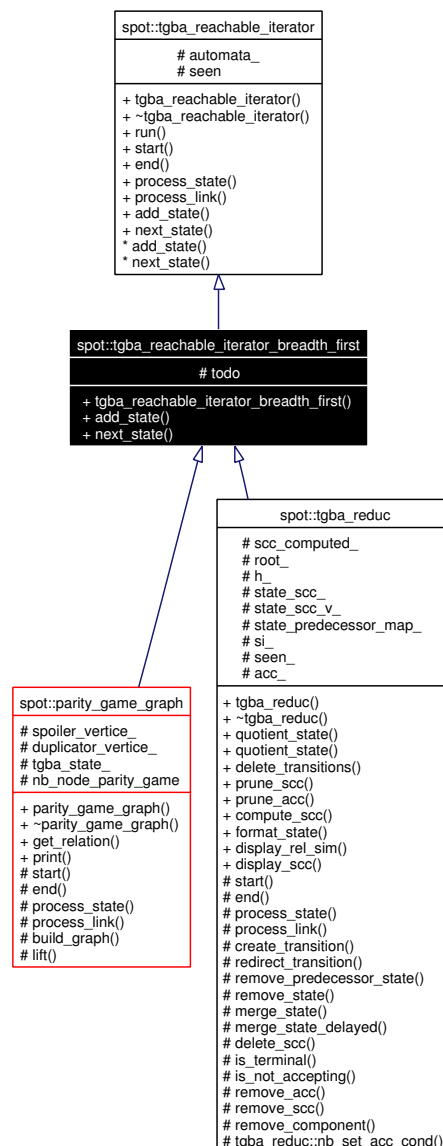
- tgbaalgos/reachiter.hh

## 11.99 spot::tgba\_reachable\_iterator\_breadth\_first Class Reference

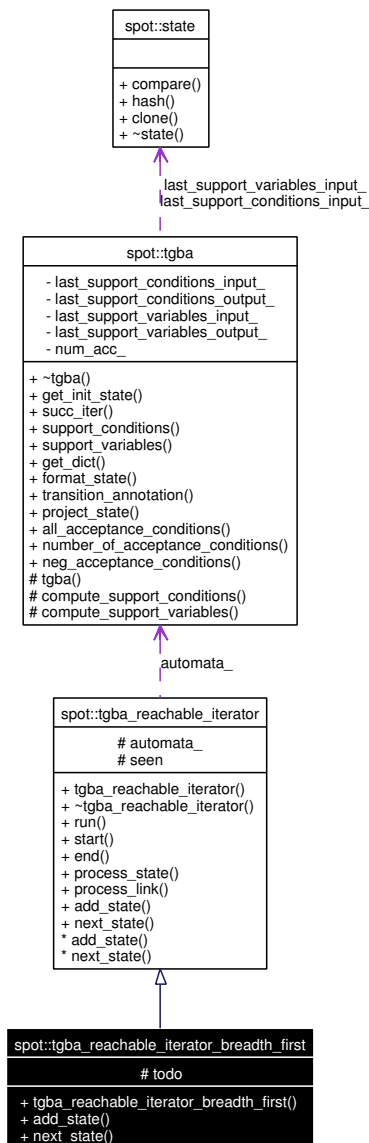
An implementation of [spot::tgba\\_reachable\\_iterator](#) that browses states breadth first.

```
#include <tgbaalgos/reachiter.hh>
```

Inheritance diagram for spot::tgba\_reachable\_iterator\_breadth\_first:



Collaboration diagram for spot::tgba\_reachable\_iterator\_breadth\_first:



## Public Member Functions

- [tgba\\_reachable\\_iterator\\_breadth\\_first](#) (const [tgba](#) \*a)
- virtual void [add\\_state](#) (const [state](#) \*s)
- virtual const [state](#) \* [next\\_state](#) ()  
*Called by [run\(\)](#) to obtain the.*
- void [run](#) ()  
*Iterate over all reachable states of a [spot::tgba](#).*
- virtual void [start](#) ()  
*Called by [run\(\)](#) before starting its iteration.*

- virtual void `end()`  
*Called by `run()` once all states have been explored.*
- virtual void `process_state` (const `state` \*s, int n, `tgba_succ_iterator` \*si)
- virtual void `process_link` (const `state` \*in\_s, int in, const `state` \*out\_s, int out, const `tgba_succ_iterator` \*si)

### Protected Types

- typedef `Sgi::hash_map< const state *, int, state_ptr_hash, state_ptr_equal >` `seen_map`

### Protected Attributes

- `std::deque< const state * >` `todo`  
*A queue of states yet to explore.*
- const `tgba` \* `automata_`  
*The `spot::tgba` to explore.*
- `seen_map` `seen`  
*States already seen.*

## 11.99.1 Detailed Description

An implementation of `spot::tgba_reachable_iterator` that browses states breadth first.

## 11.99.2 Member Typedef Documentation

**11.99.2.1** `typedef Sgi::hash_map<const state*, int, state_ptr_hash, state_ptr_equal> spot::tgba_reachable_iterator::seen_map` [protected, inherited]

Reimplemented in `spot::tgba_reduc`.

## 11.99.3 Constructor & Destructor Documentation

**11.99.3.1** `spot::tgba_reachable_iterator_breadth_first::tgba_reachable_iterator_breadth_first (const tgba * a)`

## 11.99.4 Member Function Documentation

**11.99.4.1** `virtual void spot::tgba_reachable_iterator_breadth_first::add_state (const state * s)` [virtual]

Implements `spot::tgba_reachable_iterator`.

**11.99.4.2 virtual void spot::tgba\_reachable\_iterator::end ()** [virtual, inherited]

Called by [run\(\)](#) once all states have been explored.

Reimplemented in [spot::tgba\\_reduc](#), and [spot::parity\\_game\\_graph](#).

**11.99.4.3 virtual const state\* spot::tgba\_reachable\_iterator\_breadth\_first::next\_state ()** [virtual]

Called by [run\(\)](#) to obtain the.

Implements [spot::tgba\\_reachable\\_iterator](#).

**11.99.4.4 virtual void spot::tgba\_reachable\_iterator::process\_link (const state \* in\_s, int in, const state \* out\_s, int out, const tgba\_succ\_iterator \* si)** [virtual, inherited]

Called by [run\(\)](#) to process a transition.

**Parameters:**

*in\_s* The source state

*in* The source state number.

*out\_s* The destination state

*out* The destination state number.

*si* The [spot::tgba\\_succ\\_iterator](#) positionned on the current transition.

The *in\_s* and *out\_s* states are owned by the [spot::tgba\\_reachable\\_iterator](#) instance and destroyed when the instance is destroyed.

**11.99.4.5 virtual void spot::tgba\_reachable\_iterator::process\_state (const state \* s, int n, tgba\_succ\_iterator \* si)** [virtual, inherited]

Called by [run\(\)](#) to process a state.

**Parameters:**

*s* The current state.

*n* A unique number assigned to *s*.

*si* The [spot::tgba\\_succ\\_iterator](#) for *s*.

Reimplemented in [spot::parity\\_game\\_graph](#).

**11.99.4.6 void spot::tgba\_reachable\_iterator::run ()** [inherited]

Iterate over all reachable states of a [spot::tgba](#).

This is a template method that will call [add\\_state\(\)](#), [next\\_state\(\)](#), [start\(\)](#), [end\(\)](#), [process\\_state\(\)](#), and [process\\_link\(\)](#), while it iterate over state.

**11.99.4.7 virtual void spot::tgba\_reachable\_iterator::start ()** [virtual, inherited]

Called by [run\(\)](#) before starting its iteration.

Reimplemented in [spot::tgba\\_reduc](#), and [spot::parity\\_game\\_graph](#).

### 11.99.5 Member Data Documentation

**11.99.5.1** `const tgba* spot::tgba_reachable_iterator::automata_` [protected, inherited]

The `spot::tgba` to explore.

**11.99.5.2** `seen_map spot::tgba_reachable_iterator::seen` [protected, inherited]

States already seen.

**11.99.5.3** `std::deque<const state*> spot::tgba_reachable_iterator_breadth_first::todo` [protected]

A queue of states yet to explore.

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

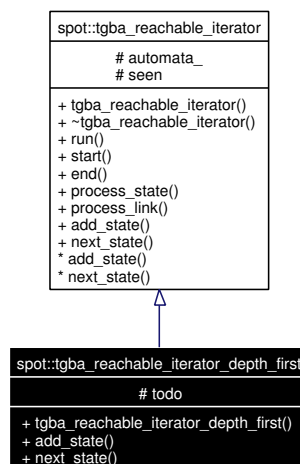
- `tgbaalgos/reachiter.hh`

## 11.100 spot::tgba\_reachable\_iterator\_depth\_first Class Reference

An implementation of `spot::tgba_reachable_iterator` that browses states depth first.

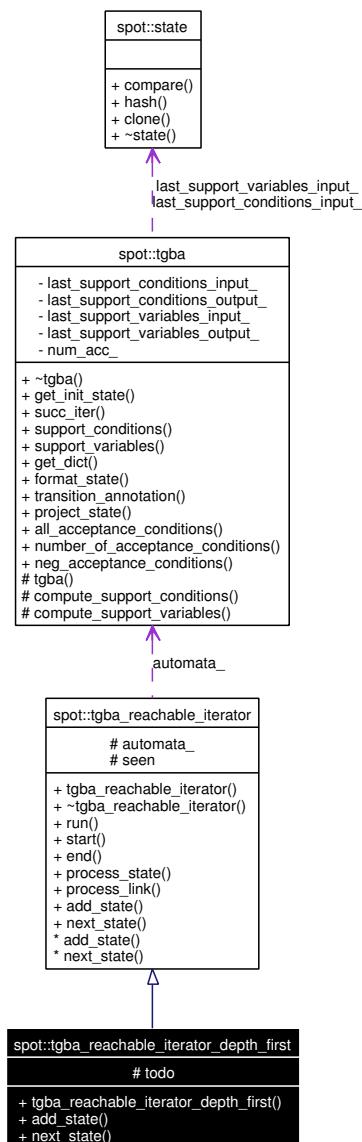
```
#include <tgbaalgos/reachiter.hh>
```

Inheritance diagram for `spot::tgba_reachable_iterator_depth_first`:



Collaboration diagram for `spot::tgba_reachable_iterator_depth_first`:





## Public Member Functions

- [tgba\\_reachable\\_iterator\\_depth\\_first](#) (const [tgba](#) \*a)
- virtual void [add\\_state](#) (const [state](#) \*s)
- virtual const [state](#) \* [next\\_state](#) ()

*Called by [run\(\)](#) to obtain the.*

- void [run](#) ()

*Iterate over all reachable states of a [spot::tgba](#).*

- virtual void [start](#) ()

*Called by [run\(\)](#) before starting its iteration.*

- virtual void [end](#) ()

Called by `run()` once all states have been explored.

- virtual void `process_state` (const `state` \*s, int n, `tgba_succ_iterator` \*si)
- virtual void `process_link` (const `state` \*in\_s, int in, const `state` \*out\_s, int out, const `tgba_succ_iterator` \*si)

### Protected Types

- typedef `Sgi::hash_map`< const `state` \*, int, `state_ptr_hash`, `state_ptr_equal` > `seen_map`

### Protected Attributes

- `std::stack`< const `state` \* > `todo`  
A stack of states yet to explore.
- const `tgba` \* `automata_`  
The `spot::tgba` to explore.
- `seen_map` `seen`  
States already seen.

#### 11.100.1 Detailed Description

An implementation of `spot::tgba_reachable_iterator` that browses states depth first.

#### 11.100.2 Member Typedef Documentation

**11.100.2.1** typedef `Sgi::hash_map`<const `state`\*, int, `state_ptr_hash`, `state_ptr_equal`> `spot::tgba_reachable_iterator::seen_map` [protected, inherited]

Reimplemented in `spot::tgba_reduc`.

#### 11.100.3 Constructor & Destructor Documentation

**11.100.3.1** `spot::tgba_reachable_iterator_depth_first::tgba_reachable_iterator_depth_first` (const `tgba` \* a)

#### 11.100.4 Member Function Documentation

**11.100.4.1** virtual void `spot::tgba_reachable_iterator_depth_first::add_state` (const `state` \* s) [virtual]

Implements `spot::tgba_reachable_iterator`.

**11.100.4.2** virtual void `spot::tgba_reachable_iterator::end` () [virtual, inherited]

Called by `run()` once all states have been explored.

Reimplemented in `spot::tgba_reduc`, and `spot::parity_game_graph`.

**11.100.4.3** `virtual const state* spot::tgba_reachable_iterator_depth_first::next_state ()` [virtual]

Called by `run()` to obtain the.

Implements `spot::tgba_reachable_iterator`.

**11.100.4.4** `virtual void spot::tgba_reachable_iterator::process_link (const state * in_s, int in, const state * out_s, int out, const tgba_succ_iterator * si)` [virtual, inherited]

Called by `run()` to process a transition.

**Parameters:**

*in\_s* The source state

*in* The source state number.

*out\_s* The destination state

*out* The destination state number.

*si* The `spot::tgba_succ_iterator` positionned on the current transition.

The *in\_s* and *out\_s* states are owned by the `spot::tgba_reachable_iterator` instance and destroyed when the instance is destroyed.

**11.100.4.5** `virtual void spot::tgba_reachable_iterator::process_state (const state * s, int n, tgba_succ_iterator * si)` [virtual, inherited]

Called by `run()` to process a state.

**Parameters:**

*s* The current state.

*n* A unique number assigned to *s*.

*si* The `spot::tgba_succ_iterator` for *s*.

Reimplemented in `spot::parity_game_graph`.

**11.100.4.6** `void spot::tgba_reachable_iterator::run ()` [inherited]

Iterate over all reachable states of a `spot::tgba`.

This is a template method that will call `add_state()`, `next_state()`, `start()`, `end()`, `process_state()`, and `process_link()`, while it iterate over state.

**11.100.4.7** `virtual void spot::tgba_reachable_iterator::start ()` [virtual, inherited]

Called by `run()` before starting its iteration.

Reimplemented in `spot::tgba_reduc`, and `spot::parity_game_graph`.

## 11.100.5 Member Data Documentation

**11.100.5.1** `const tgba* spot::tgba_reachable_iterator::automata_` [protected, inherited]

The `spot::tgba` to explore.

**11.100.5.2** [seen\\_map spot::tgba\\_reachable\\_iterator::seen](#) [protected, inherited]

States already seen.

**11.100.5.3** `std::stack<const state\*>` [spot::tgba\\_reachable\\_iterator\\_depth\\_first::todo](#)  
[protected]

A stack of states yet to explore.

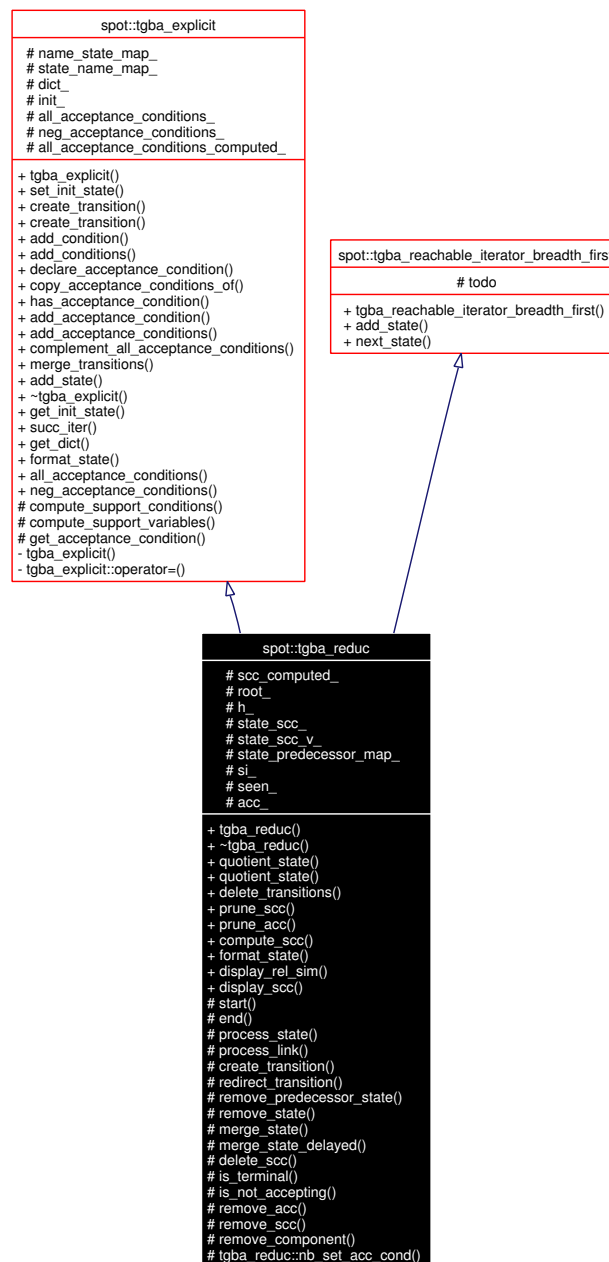
The documentation for this class was generated from the following file:

- [tgbaalgos/reachiter.hh](#)

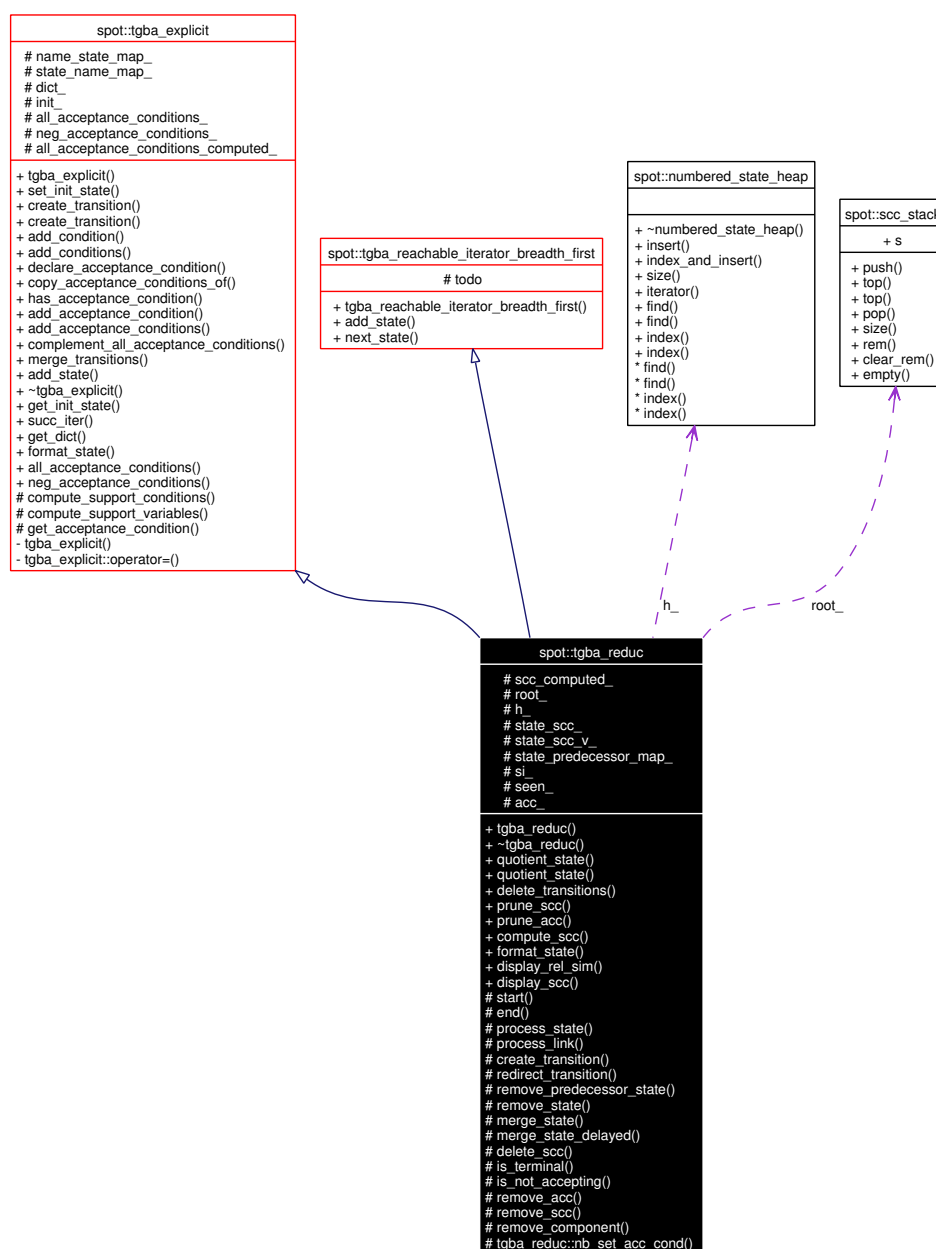
**11.101** spot::tgba\_reduc Class Reference

```
#include <tgba/tgbareduc.hh>
```

Inheritance diagram for spot::tgba\_reduc:



Collaboration diagram for spot::tgba\_reduc:



## Public Types

- typedef std::list< transition \* > [state](#)

## Public Member Functions

- [tgba\\_reduc](#) (const [tgba](#) \*a, const [numbered\\_state\\_heap\\_factory](#) \*nshf=numbered\_state\_heap\_hash\_map\_factory::instance())
- [~tgba\\_reduc](#) ()
- void [quotient\\_state](#) (direct\_simulation\_relation \*rel)
- void [quotient\\_state](#) (delayed\_simulation\_relation \*rel)

- void [delete\\_transitions](#) ([simulation\\_relation](#) \*rel)  
*Delete some transitions with help of a simulation relation.*
- void [prune\\_scc](#) ()  
*Remove all state which not lead to an accepting cycle.*
- void [prune\\_acc](#) ()  
*Remove some useless accepting condition.*
- void [compute\\_scc](#) ()  
*Compute the maximal SCC of the automata.*
- virtual std::string [format\\_state](#) (const [spot::state](#) \*state) const  
*Add the SCC index to the display of the state state.*
- void [display\\_rel\\_sim](#) ([simulation\\_relation](#) \*rel, std::ostream &os)
- void [display\\_scc](#) (std::ostream &os)
- [state](#) \* [set\\_init\\_state](#) (const std::string &state)
- transition \* [create\\_transition](#) (const std::string &source, const std::string &dest)
- transition \* [create\\_transition](#) ([state](#) \*source, const [state](#) \*dest)
- void [add\\_condition](#) (transition \*t, const [ltl::formula](#) \*f)
- void [add\\_conditions](#) (transition \*t, bdd f)  
*This assumes that all variables in f are known from dict.*
- void [declare\\_acceptance\\_condition](#) (const [ltl::formula](#) \*f)
- void [copy\\_acceptance\\_conditions\\_of](#) (const [tgba](#) \*a)  
*Copy the acceptance conditions of a tgba.*
- bool [has\\_acceptance\\_condition](#) (const [ltl::formula](#) \*f) const
- void [add\\_acceptance\\_condition](#) (transition \*t, const [ltl::formula](#) \*f)
- void [add\\_acceptance\\_conditions](#) (transition \*t, bdd f)  
*This assumes that all acceptance conditions in f are known from dict.*
- void [complement\\_all\\_acceptance\\_conditions](#) ()
- void [merge\\_transitions](#) ()
- [state](#) \* [add\\_state](#) (const std::string &name)
- virtual [spot::state](#) \* [get\\_init\\_state](#) () const  
*Get the initial state of the automaton.*
- virtual [tgba\\_succ\\_iterator](#) \* [succ\\_iter](#) (const [spot::state](#) \*local\_state, const [spot::state](#) \*global\_state=0, const [tgba](#) \*global\_automaton=0) const
- virtual [tgba\\_succ\\_iterator](#) \* [succ\\_iter](#) (const [state](#) \*local\_state, const [state](#) \*global\_state=0, const [tgba](#) \*global\_automaton=0) const =0  
*Get an iterator over the successors of local\_state.*
- virtual [bdd\\_dict](#) \* [get\\_dict](#) () const  
*Get the dictionary associated to the automaton.*
- virtual std::string [format\\_state](#) (const [state](#) \*state) const =0  
*Format the state as a string for printing.*

- virtual bdd [all\\_acceptance\\_conditions](#) () const  
*Return the set of all acceptance conditions used by this automaton.*
- virtual bdd [neg\\_acceptance\\_conditions](#) () const  
*Return the conjunction of all negated acceptance variables.*
- bdd [support\\_conditions](#) (const [state](#) \*state) const  
*Get a formula that must hold whatever successor is taken.*
- bdd [support\\_variables](#) (const [state](#) \*state) const  
*Get the conjunctions of variables tested by the outgoing transitions of state.*
- virtual std::string [transition\\_annotation](#) (const [tgba\\_succ\\_iterator](#) \*t) const  
*Return a possible annotation for the transition pointed to by the iterator.*
- virtual [state](#) \* [project\\_state](#) (const [state](#) \*s, const [tgba](#) \*t) const  
*Project a state on an automaton.*
- virtual int [number\\_of\\_acceptance\\_conditions](#) () const  
*The number of acceptance conditions.*
- virtual void [add\\_state](#) (const [state](#) \*s)
- virtual const [state](#) \* [next\\_state](#) ()  
*Called by [run\(\)](#) to obtain the.*
- void [run](#) ()  
*Iterate over all reachable states of a [spot::tgba](#).*
- virtual void [process\\_state](#) (const [state](#) \*s, int n, [tgba\\_succ\\_iterator](#) \*si)
- virtual void [process\\_link](#) (const [state](#) \*in\_s, int in, const [state](#) \*out\_s, int out, const [tgba\\_succ\\_iterator](#) \*si)

## Protected Types

- typedef Sgi::hash\_map< const [tgba\\_explicit::state](#) \*, std::list< [state](#) \* > \*, ptr\_hash< [tgba\\_explicit::state](#) > > [sp\\_map](#)
- typedef Sgi::hash\_map< const [spot::state](#) \*, int, state\_ptr\_hash, state\_ptr\_equal > [seen\\_map](#)
- typedef Sgi::hash\_map< const std::string, [tgba\\_explicit::state](#) \*, string\_hash > [ns\\_map](#)
- typedef Sgi::hash\_map< const [tgba\\_explicit::state](#) \*, std::string, ptr\_hash< [tgba\\_explicit::state](#) > > [sn\\_map](#)

## Protected Member Functions

- void [start](#) ()  
*Called by [run\(\)](#) before starting its iteration.*
- void [end](#) ()  
*Called by [run\(\)](#) once all states have been explored.*



- void [process\\_state](#) (const [spot::state](#) \*s, int n, [tgba\\_succ\\_iterator](#) \*si)
- void [process\\_link](#) (int in, int out, const [tgba\\_succ\\_iterator](#) \*si)
- transition \* [create\\_transition](#) (const [spot::state](#) \*source, const [spot::state](#) \*dest)

*Create a transition using two state of a TGBA.*

- void [redirect\\_transition](#) (const [spot::state](#) \*s, const [spot::state](#) \*simul)
- void [remove\\_predecessor\\_state](#) (const [state](#) \*s, const [state](#) \*p)

*Remove p of the predecessor of s.*

- void [remove\\_state](#) (const [spot::state](#) \*s)
- void [merge\\_state](#) (const [spot::state](#) \*s1, const [spot::state](#) \*s2)
- void [merge\\_state\\_delayed](#) (const [spot::state](#) \*s1, const [spot::state](#) \*s2)
- void [delete\\_scc](#) ()
- bool [is\\_terminal](#) (const [spot::state](#) \*s, int n=-1)
- bool [is\\_not\\_accepting](#) (const [spot::state](#) \*s, int n=-1)
- void [remove\\_acc](#) (const [spot::state](#) \*s)
- void [remove\\_scc](#) ([spot::state](#) \*s)

*Remove all the state which belong to the same scc that s.*

- void [remove\\_component](#) (const [spot::state](#) \*from)

*For compute\_scc.*

- int [tgba\\_reduc::nb\\_set\\_acc\\_cond](#) () const
- virtual bdd [compute\\_support\\_conditions](#) (const [spot::state](#) \*state) const
- virtual bdd [compute\\_support\\_conditions](#) (const [state](#) \*state) const =0

*Do the actual computation of tgba::support\_conditions().*

- virtual bdd [compute\\_support\\_variables](#) (const [spot::state](#) \*state) const
- virtual bdd [compute\\_support\\_variables](#) (const [state](#) \*state) const =0

*Do the actual computation of tgba::support\_variables().*

- bdd [get\\_acceptance\\_condition](#) (const [ltl::formula](#) \*f)

## Protected Attributes

- bool [scc\\_computed\\_](#)
- [scc\\_stack](#) root\_
- [numbered\\_state\\_heap](#) \* h\_
- std::stack< const [spot::state](#) \* > [state\\_scc\\_](#)
- Sgi::hash\_map< int, const [spot::state](#) \* > [state\\_scc\\_v\\_](#)
- [sp\\_map](#) [state\\_predecessor\\_map\\_](#)
- [seen\\_map](#) si\_
- [seen\\_map](#) \* seen\_
- bdd [acc\\_](#)
- [ns\\_map](#) [name\\_state\\_map\\_](#)
- [sn\\_map](#) [state\\_name\\_map\\_](#)
- bdd\_dict \* dict\_
- [tgba\\_explicit::state](#) \* init\_
- bdd [all\\_acceptance\\_conditions\\_](#)
- bdd [neg\\_acceptance\\_conditions\\_](#)

- bool [all\\_acceptance\\_conditions\\_computed\\_](#)
- `std::deque< const state * >` [todo](#)  
*A queue of states yet to explore.*
- const [tgba](#) \* [automata\\_](#)  
*The [spot::tgba](#) to explore.*
- [seen\\_map](#) `seen`  
*States already seen.*

### 11.101.1 Detailed Description

Explicit automata used in reductions.

### 11.101.2 Member Typedef Documentation

**11.101.2.1** `typedef Sgi::hash_map<const std::string, tgba\_explicit::state\*, string\_hash>`  
[spot::tgba\\_explicit::ns\\_map](#) [protected, inherited]

**11.101.2.2** `typedef Sgi::hash_map<const spot::state\*, int, state\_ptr\_hash, state\_ptr\_equal>`  
[spot::tgba\\_reduc::seen\\_map](#) [protected]

Reimplemented from [spot::tgba\\_reachable\\_iterator](#).

**11.101.2.3** `typedef Sgi::hash_map<const tgba\_explicit::state\*, std::string, ptr\_hash<tgba\_explicit::state>>`  
[spot::tgba\\_explicit::sn\\_map](#) [protected, inherited]

**11.101.2.4** `typedef Sgi::hash_map<const tgba\_explicit::state\*, std::list<state\*>*, ptr\_hash<tgba\_explicit::state>>`  
[spot::tgba\\_reduc::sp\\_map](#) [protected]

**11.101.2.5** `typedef std::list<transition*> spot::tgba\_explicit::state` [inherited]

### 11.101.3 Constructor & Destructor Documentation

**11.101.3.1** `spot::tgba_reduc::tgba_reduc (const tgba * a, const numbered\_state\_heap\_factory * nshf)`  
*nshf* = `numbered_state_heap_hash_map_factory::instance()`

**11.101.3.2** `spot::tgba_reduc::~~tgba_reduc ()`

### 11.101.4 Member Function Documentation

**11.101.4.1** `void spot::tgba_explicit::add_acceptance_condition (transition * t, const ltl::formula * f)` [inherited]

**11.101.4.2** `void spot::tgba_explicit::add_acceptance_conditions (transition * t, bdd f)` [inherited]

This assumes that all acceptance conditions in  $f$  are known from dict.

**11.101.4.3** `void spot::tgba_explicit::add_condition (transition * t, const ltl::formula * f)` [inherited]

**11.101.4.4** `void spot::tgba_explicit::add_conditions (transition * t, bdd f)` [inherited]

This assumes that all variables in  $f$  are known from dict.

**11.101.4.5** `virtual void spot::tgba_reachable_iterator_breadth_first::add_state (const state * s)` [virtual, inherited]

Implements [spot::tgba\\_reachable\\_iterator](#).

**11.101.4.6** `state* spot::tgba_explicit::add_state (const std::string & name)` [inherited]

Return the [tgba\\_explicit::state](#) for  $name$ , creating the state if it does not exist.

**11.101.4.7** `virtual bdd spot::tgba_explicit::all_acceptance_conditions () const` [virtual, inherited]

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::tgba](#).

**11.101.4.8** `void spot::tgba_explicit::complement_all_acceptance_conditions ()` [inherited]

**11.101.4.9** `void spot::tgba_reduc::compute_scc ()`

Compute the maximal SCC of the automata.

**11.101.4.10** `virtual bdd spot::tgba::compute_support_conditions (const state * state) const` [protected, pure virtual, inherited]

Do the actual computation of [tgba::support\\_conditions\(\)](#).

Implemented in [spot::tgba\\_bdd\\_concrete](#), [spot::tgba\\_product](#), and [spot::tgba\\_tba\\_proxy](#).

**11.101.4.11** `virtual bdd spot::tgba_explicit::compute_support_conditions (const spot::state * state) const` [protected, virtual, inherited]

**11.101.4.12** `virtual bdd spot::tgba::compute_support_variables (const state * state) const` [protected, pure virtual, inherited]

Do the actual computation of [tgba::support\\_variables\(\)](#).

Implemented in [spot::tgba\\_bdd\\_concrete](#), [spot::tgba\\_product](#), and [spot::tgba\\_tba\\_proxy](#).

**11.101.4.13** virtual bdd spot::tgba\_explicit::compute\_support\_variables (const spot::state \* state) const [protected, virtual, inherited]

**11.101.4.14** void spot::tgba\_explicit::copy\_acceptance\_conditions\_of (const tgba \* a) [inherited]

Copy the acceptance conditions of a tgba.

If used, this function should be called before creating any transition.

**11.101.4.15** transition\* spot::tgba\_explicit::create\_transition (state \* source, const state \* dest) [inherited]

**11.101.4.16** transition\* spot::tgba\_explicit::create\_transition (const std::string & source, const std::string & dest) [inherited]

**11.101.4.17** transition\* spot::tgba\_reduc::create\_transition (const spot::state \* source, const spot::state \* dest) [protected]

Create a transition using two state of a TGBA.

**11.101.4.18** void spot::tgba\_explicit::declare\_acceptance\_condition (const ltl::formula \* f) [inherited]

**11.101.4.19** void spot::tgba\_reduc::delete\_scc () [protected]

Remove all the scc which are terminal and doesn't contains all the acceptance conditions.

**11.101.4.20** void spot::tgba\_reduc::delete\_transitions (simulation\_relation \* rel)

Delete some transitions with help of a simulation relation.

**11.101.4.21** void spot::tgba\_reduc::display\_rel\_sim (simulation\_relation \* rel, std::ostream & os)

**11.101.4.22** void spot::tgba\_reduc::display\_scc (std::ostream & os)

**11.101.4.23** void spot::tgba\_reduc::end () [protected, virtual]

Called by run() once all states have been explored.

Reimplemented from spot::tgba\_reachable\_iterator.

**11.101.4.24** virtual std::string spot::tgba::format\_state (const state \* state) const [pure virtual, inherited]

Format the state as a string for printing.

This formatting is the responsibility of the automata who owns the state.

Implemented in spot::tgba\_bdd\_concrete, spot::tgba\_product, and spot::tgba\_tba\_proxy.

**11.101.4.25** `virtual std::string spot::tgba_reduc::format_state (const spot::state * state) const` [virtual]

Add the SCC index to the display of the state *state*.

Reimplemented from [spot::tgba\\_explicit](#).

**11.101.4.26** `bdd spot::tgba_explicit::get_acceptance_condition (const ltl::formula * f)` [protected, inherited]

**11.101.4.27** `virtual bdd_dict* spot::tgba_explicit::get_dict () const` [virtual, inherited]

Get the dictionary associated to the automaton.

State are represented as BDDs. The dictionary allows to map BDD variables back to formulae, and vice versa. This is useful when dealing with several automata (which may use the same BDD variable for different formula), or simply when printing.

Implements [spot::tgba](#).

**11.101.4.28** `virtual spot::state* spot::tgba_explicit::get_init_state () const` [virtual, inherited]

Get the initial state of the automaton.

The state has been allocated with new. It is the responsibility of the caller to delete it when no longer needed.

Implements [spot::tgba](#).

**11.101.4.29** `bool spot::tgba_explicit::has_acceptance_condition (const ltl::formula * f) const` [inherited]

**11.101.4.30** `bool spot::tgba_reduc::is_not_accepting (const spot::state * s, int n = -1)` [protected]

**11.101.4.31** `bool spot::tgba_reduc::is_terminal (const spot::state * s, int n = -1)` [protected]

Return true if the scc which contains *s* is an fixed-formula alpha-ball. this is explain in

```
@InProceedings{ etessami.00.concur,
  author    = {Kousha Etessami and Gerard J. Holzmann},
  title     = {Optimizing {B\"u}chi Automata},
  booktitle = {Proceedings of the 11th International Conference on
    Concurrency Theory (Concur'2000)},
  pages     = {153--167},
  year      = {2000},
  editor    = {C. Palamidessi},
  volume    = {1877},
  series    = {Lecture Notes in Computer Science},
  publisher = {Springer-Verlag}
}
```

**11.101.4.32** void spot::tgba\_reduc::merge\_state (const spot::state \* s1, const spot::state \* s2) [protected]

Redirect all transition leading to s1 to s2. Note that we can do the reverse because s1 and s2 belong to a co-simulate relation.

**11.101.4.33** void spot::tgba\_reduc::merge\_state\_delayed (const spot::state \* s1, const spot::state \* s2) [protected]

Redirect all transition leading to s1 to s2. Note that we can do the reverse because s1 and s2 belong to a co-simulate relation.

**11.101.4.34** void spot::tgba\_explicit::merge\_transitions () [inherited]

**11.101.4.35** virtual bdd spot::tgba\_explicit::neg\_acceptance\_conditions () const [virtual, inherited]

Return the conjunction of all negated acceptance variables.

For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a]&!Acc[b]&!Acc[c]`.

This is useful when making products: each operand's condition set should be augmented with the `neg_acceptance_conditions()` of the other operand.

Implements `spot::tgba`.

**11.101.4.36** virtual const state\* spot::tgba\_reachable\_iterator\_breadth\_first::next\_state () [virtual, inherited]

Called by `run()` to obtain the.

Implements `spot::tgba_reachable_iterator`.

**11.101.4.37** virtual int spot::tgba::number\_of\_acceptance\_conditions () const [virtual, inherited]

The number of acceptance conditions.

**11.101.4.38** virtual void spot::tgba\_reachable\_iterator::process\_link (const state \* in\_s, int in, const state \* out\_s, int out, const tgba\_succ\_iterator \* si) [virtual, inherited]

Called by `run()` to process a transition.

#### Parameters:

*in\_s* The source state

*in* The source state number.

*out\_s* The destination state

*out* The destination state number.

*si* The `spot::tgba_succ_iterator` positionned on the current transition.

The `in_s` and `out_s` states are owned by the `spot::tgba_reachable_iterator` instance and destroyed when the instance is destroyed.

**11.101.4.39** void spot::tgba\_reduc::process\_link (int *in*, int *out*, const tgba\_succ\_iterator \* *si*) [protected]

**11.101.4.40** virtual void spot::tgba\_reachable\_iterator::process\_state (const state \* *s*, int *n*, tgba\_succ\_iterator \* *si*) [virtual, inherited]

Called by run() to process a state.

**Parameters:**

- s* The current state.
- n* A unique number assigned to *s*.
- si* The spot::tgba\_succ\_iterator for *s*.

Reimplemented in spot::parity\_game\_graph.

**11.101.4.41** void spot::tgba\_reduc::process\_state (const spot::state \* *s*, int *n*, tgba\_succ\_iterator \* *si*) [protected]

**11.101.4.42** virtual state\* spot::tgba::project\_state (const state \* *s*, const tgba \* *t*) const [virtual, inherited]

Project a state on an automaton.

This converts *s*, into that corresponding spot::state for *t*. This is useful when you have the state of a product, and want restrict this state to a specific automata occuring in the product.

It goes without saying that *s* and *t* should be compatible (i.e., *s* is a state of *t*).

**Returns:**

- 0 if the projection fails (*s* is unrelated to *t*), or a new state\* (the projected state) that must be deleted by the caller.

Reimplemented in spot::tgba\_product, and spot::tgba\_tba\_proxy.

**11.101.4.43** void spot::tgba\_reduc::prune\_acc ()

Remove some useless accepting condition.

**11.101.4.44** void spot::tgba\_reduc::prune\_scc ()

Remove all state which not lead to an accepting cycle.

**11.101.4.45** void spot::tgba\_reduc::quotient\_state (delayed\_simulation\_relation \* *rel*)

Build the quotient automata. Call this method when use to a delayed simulation relation.

**11.101.4.46** void spot::tgba\_reduc::quotient\_state (direct\_simulation\_relation \* *rel*)

Reduce the automata using a relation simulation Do not call this method with a delayed simulation relation.

**11.101.4.47** void spot::tgba\_reduc::redirect\_transition (const spot::state \* s, const spot::state \* simul) [protected]

Remove all the transition from the state q, predecessor of both s and simul, which can be removed.

**11.101.4.48** void spot::tgba\_reduc::remove\_acc (const spot::state \* s) [protected]

If a scc maximal do not contains all the accepting condition we can remove all the accepting condition in this scc.

**11.101.4.49** void spot::tgba\_reduc::remove\_component (const spot::state \* from) [protected]

For compute\_scc.

**11.101.4.50** void spot::tgba\_reduc::remove\_predecessor\_state (const state \* s, const state \* p) [protected]

Remove p of the predecessor of s.

**11.101.4.51** void spot::tgba\_reduc::remove\_scc (spot::state \* s) [protected]

Remove all the state which belong to the same scc that s.

**11.101.4.52** void spot::tgba\_reduc::remove\_state (const spot::state \* s) [protected]

Remove all the transition leading to s. s is then unreachable and can be consider as remove.

**11.101.4.53** void spot::tgba\_reachable\_iterator::run () [inherited]

Iterate over all reachable states of a spot::tgba.

This is a template method that will call add\_state(), next\_state(), start(), end(), process\_state(), and process\_link(), while it iterate over state.

**11.101.4.54** state\* spot::tgba\_explicit::set\_init\_state (const std::string & state) [inherited]

**11.101.4.55** void spot::tgba\_reduc::start () [protected, virtual]

Called by run() before starting its iteration.

Reimplemented from spot::tgba\_reachable\_iterator.

**11.101.4.56** virtual tgba\_succ\_iterator\* spot::tgba::succ\_iter (const state \* local\_state, const state \* global\_state = 0, const tgba \* global\_automaton = 0) const [pure virtual, inherited]

Get an iterator over the successors of local\_state.

The iterator has been allocated with new. It is the responsibility of the caller to delete it when no longer needed.

During synchornized products, additional informations are passed about the entire product and its state. Recall that products can be nested, forming a tree of spot::tgba where most values are computed on demand. global\_automaton designate the root spot::tgba, and global\_state its state. This two objects can be used by succ\_iter() to restrict the set of successors to compute.



**Parameters:**

**local\_state** The state whose successors are to be explored. This pointer is not adopted in any way by `succ_iter`, and it is still the caller's responsibility to delete it when appropriate (this can be done during the lifetime of the iterator).

**global\_state** In a product, the state of the global product automaton. Otherwise, 0. Like `local_state`, `global_state` is not adopted by `succ_iter`.

**global\_automaton** In a product, the global product automaton. Otherwise, 0.

Implemented in `spot::tgba_bdd_concrete`, `spot::tgba_product`, and `spot::tgba_tba_proxy`.

**11.101.4.57** `virtual tgba_succ_iterator* spot::tgba_explicit::succ_iter (const spot::state * local_state, const spot::state * global_state = 0, const tgba * global_automaton = 0) const` [virtual, inherited]

**11.101.4.58** `bdd spot::tgba::support_conditions (const state * state) const` [inherited]

Get a formula that must hold whatever successor is taken.

**Returns:**

A formula which must be verified for all successors of `state`.

This can be as simple as `bddtrue`, or more completely the disjunction of the condition of all successors. This is used as an hint by `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_conditions()`, this function is just a wrapper that will cache the last return value for efficiency.

**11.101.4.59** `bdd spot::tgba::support_variables (const state * state) const` [inherited]

Get the conjunctions of variables tested by the outgoing transitions of `state`.

All variables tested by outgoing transitions must be returned. This is mandatory.

This is used as an hint by some `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_variables()`, this function is just a wrapper that will cache the last return value for efficiency.

**11.101.4.60** `int spot::tgba_reduc::tgba_reduc::nb_set_acc_cond () const` [protected]

**11.101.4.61** `virtual std::string spot::tgba::transition_annotation (const tgba_succ_iterator * t) const` [virtual, inherited]

Return a possible annotation for the transition pointed to by the iterator.

Implementing this function is optional; the default annotation is the empty string.

**Parameters:**

**t** a non-done `tgba_succ_iterator` for this automata

Reimplemented in `spot::tgba_product`, and `spot::tgba_tba_proxy`.

### 11.101.5 Member Data Documentation

11.101.5.1 **bdd** [spot::tgba\\_reduc::acc\\_](#) [protected]

11.101.5.2 **bdd** [spot::tgba\\_explicit::all\\_acceptance\\_conditions\\_](#) [mutable, protected, inherited]

11.101.5.3 **bool** [spot::tgba\\_explicit::all\\_acceptance\\_conditions\\_computed\\_](#) [mutable, protected, inherited]

11.101.5.4 **const tgba\*** [spot::tgba\\_reachable\\_iterator::automata\\_](#) [protected, inherited]

The [spot::tgba](#) to explore.

11.101.5.5 **bdd\_dict\*** [spot::tgba\\_explicit::dict\\_](#) [protected, inherited]

11.101.5.6 **numbered\_state\_heap\*** [spot::tgba\\_reduc::h\\_](#) [protected]

11.101.5.7 **tgba\_explicit::state\*** [spot::tgba\\_explicit::init\\_](#) [protected, inherited]

11.101.5.8 **ns\_map** [spot::tgba\\_explicit::name\\_state\\_map\\_](#) [protected, inherited]

11.101.5.9 **bdd** [spot::tgba\\_explicit::neg\\_acceptance\\_conditions\\_](#) [protected, inherited]

11.101.5.10 **scc\_stack** [spot::tgba\\_reduc::root\\_](#) [protected]

11.101.5.11 **bool** [spot::tgba\\_reduc::scc\\_computed\\_](#) [protected]

11.101.5.12 **seen\_map** [spot::tgba\\_reachable\\_iterator::seen](#) [protected, inherited]

States already seen.

11.101.5.13 **seen\_map\*** [spot::tgba\\_reduc::seen\\_](#) [protected]

11.101.5.14 **seen\_map** [spot::tgba\\_reduc::si\\_](#) [protected]

11.101.5.15 **sn\_map** [spot::tgba\\_explicit::state\\_name\\_map\\_](#) [protected, inherited]

11.101.5.16 **sp\_map** [spot::tgba\\_reduc::state\\_predecessor\\_map\\_](#) [protected]

11.101.5.17 **std::stack<const spot::state\*>** [spot::tgba\\_reduc::state\\_scc\\_](#) [protected]

**11.101.5.18** Sgi::hash\_map<int, const spot::state\*> spot::tgba\_reduc::state\_scc\_v\_-  
[protected]

**11.101.5.19** std::deque<const state\*> spot::tgba\_reachable\_iterator\_breadth\_first::todo  
[protected, inherited]

A queue of states yet to explore.

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

- [tgba/tgbareduc.hh](#)

## 11.102 spot::tgba\_run Struct Reference

An accepted run, for a tgba.

```
#include <tgbaalgos/emptiness.hh>
```

### Public Types

- typedef std::list< [step](#) > [steps](#)

### Public Member Functions

- [~tgba\\_run](#) ()
- [tgba\\_run](#) ()
- [tgba\\_run](#) (const [tgba\\_run](#) &run)
- [tgba\\_run](#) & [operator=](#) (const [tgba\\_run](#) &run)

### Public Attributes

- [steps](#) prefix
- [steps](#) cycle

### Classes

- struct [step](#)

#### 11.102.1 Detailed Description

An accepted run, for a tgba.

#### 11.102.2 Member Typedef Documentation

**11.102.2.1** typedef std::list<[step](#)> [spot::tgba\\_run::steps](#)

#### 11.102.3 Constructor & Destructor Documentation

**11.102.3.1** [spot::tgba\\_run::~~tgba\\_run](#) ()

11.102.3.2 spot::tgba\_run::tgba\_run() [inline]

11.102.3.3 spot::tgba\_run::tgba\_run(const tgba\_run &run)

#### 11.102.4 Member Function Documentation

11.102.4.1 tgba\_run& spot::tgba\_run::operator=(const tgba\_run &run)

#### 11.102.5 Member Data Documentation

11.102.5.1 steps spot::tgba\_run::cycle

11.102.5.2 steps spot::tgba\_run::prefix

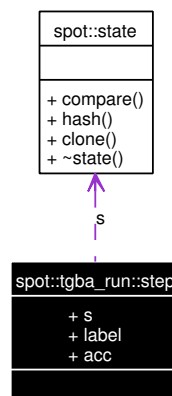
The documentation for this struct was generated from the following file:

- tgbaalgos/emptiness.hh

### 11.103 spot::tgba\_run::step Struct Reference

```
#include <tgbaalgos/emptiness.hh>
```

Collaboration diagram for spot::tgba\_run::step:



#### Public Attributes

- const state \* s
- bdd label
- bdd acc

#### 11.103.1 Member Data Documentation

11.103.1.1 bdd spot::tgba\_run::step::acc

11.103.1.2 bdd [spot::tgba\\_run::step::label](#)11.103.1.3 const [state\\*](#) [spot::tgba\\_run::step::s](#)

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

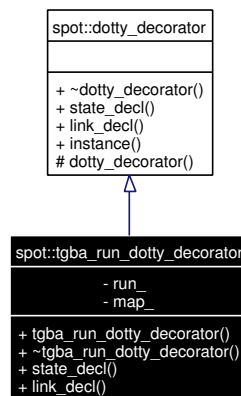
- [tgbaalgos/emptiness.hh](#)

## 11.104 spot::tgba\_run\_dotty\_decorator Class Reference

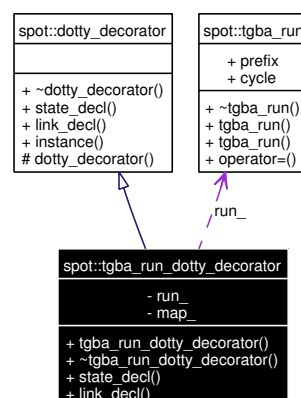
Highlight a [spot::tgba\\_run](#) on a [spot::tgba](#).

```
#include <tgbaalgos/rundotdec.hh>
```

Inheritance diagram for [spot::tgba\\_run\\_dotty\\_decorator](#):



Collaboration diagram for [spot::tgba\\_run\\_dotty\\_decorator](#):



## Public Member Functions

- [tgba\\_run\\_dotty\\_decorator](#) (const [tgba\\_run](#) \*run)
- virtual [~tgba\\_run\\_dotty\\_decorator](#) ()

- virtual std::string [state\\_decl](#) (const [tgba](#) \*a, const [state](#) \*s, int n, [tgba\\_succ\\_iterator](#) \*si, const std::string &label)

*Compute the style of a state.*

- virtual std::string [link\\_decl](#) (const [tgba](#) \*a, const [state](#) \*in\_s, int in, const [state](#) \*out\_s, int out, const [tgba\\_succ\\_iterator](#) \*si, const std::string &label)

*Compute the style of a link.*

### Static Public Member Functions

- static [dotty\\_decorator](#) \* [instance](#) ()

*Get the unique instance of the default [dotty\\_decorator](#).*

### Private Types

- typedef std::pair< [tgba\\_run::steps::const\\_iterator](#), int > [step\\_num](#)
- typedef std::list< [step\\_num](#) > [step\\_set](#)
- typedef std::map< const [state](#) \*, std::pair< [step\\_set](#), [step\\_set](#) >, [spot::state\\_ptr\\_less\\_than](#) > [step\\_map](#)

### Private Attributes

- const [tgba\\_run](#) \* [run\\_](#)
- [step\\_map](#) [map\\_](#)

#### 11.104.1 Detailed Description

Highlight a [spot::tgba\\_run](#) on a [spot::tgba](#).

An instance of this class can be passed to [spot::dotty\\_reachable](#).

#### 11.104.2 Member Typedef Documentation

**11.104.2.1** typedef std::map<const [state](#)\*, std::pair<[step\\_set](#), [step\\_set](#)>, [spot::state\\_ptr\\_less\\_than](#)> [spot::tgba\\_run\\_dotty\\_decorator::step\\_map](#) [private]

**11.104.2.2** typedef std::pair<[tgba\\_run::steps::const\\_iterator](#), int> [spot::tgba\\_run\\_dotty\\_decorator::step\\_num](#) [private]

**11.104.2.3** typedef std::list<[step\\_num](#)> [spot::tgba\\_run\\_dotty\\_decorator::step\\_set](#) [private]

#### 11.104.3 Constructor & Destructor Documentation

**11.104.3.1** [spot::tgba\\_run\\_dotty\\_decorator::tgba\\_run\\_dotty\\_decorator](#) (const [tgba\\_run](#) \* [run](#))

**11.104.3.2** virtual spot::tgba\_run\_dotty\_decorator::~~tgba\_run\_dotty\_decorator () [virtual]

#### 11.104.4 Member Function Documentation

**11.104.4.1** static dotty\_decorator\* spot::dotty\_decorator::instance () [static, inherited]

Get the unique instance of the default dotty\_decorator.

**11.104.4.2** virtual std::string spot::tgba\_run\_dotty\_decorator::link\_decl (const tgba \* a, const state \* in\_s, int in, const state \* out\_s, int out, const tgba\_succ\_iterator \* si, const std::string & label) [virtual]

Compute the style of a link.

This function should output a string of the form [label="foo", style=bar, ...]. The default implementation will simply output [label="LABEL" ] with LABEL replaced by the value of *label*.

##### Parameters:

- a* the automaton being drawn
- in\_s* the source state of the transition being drawn (owned by the caller)
- in* the unique number associated to *in\_s*
- out\_s* the destination state of the transition being drawn (owned by the caller)
- out* the unique number associated to *out\_s*
- si* an iterator over the successors of *in\_s*, pointing to the current transition (owned by the caller and cannot be iterated)
- label* the computed name of this state

Reimplemented from spot::dotty\_decorator.

**11.104.4.3** virtual std::string spot::tgba\_run\_dotty\_decorator::state\_decl (const tgba \* a, const state \* s, int n, tgba\_succ\_iterator \* si, const std::string & label) [virtual]

Compute the style of a state.

This function should output a string of the form [label="foo", style=bar, ...]. The default implementation will simply output [label="LABEL" ] with LABEL replaced by the value of *label*.

##### Parameters:

- a* the automaton being drawn
- s* the state being drawn (owned by the caller)
- n* a unique number for this state
- si* an iterator over the successors of this state (owned by the caller, but can be freely iterated)
- label* the computed name of this state

Reimplemented from spot::dotty\_decorator.

#### 11.104.5 Member Data Documentation

**11.104.5.1** step\_map spot::tgba\_run\_dotty\_decorator::map\_ [private]

## 11.104.5.2 const tgba\_run\* spot::tgba\_run\_dotty\_decorator::run\_ [private]

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

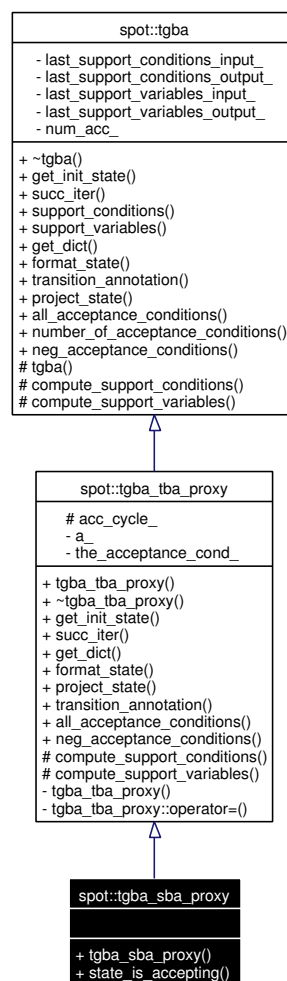
- tgbaalgos/rundotdec.hh

## 11.105 spot::tgba\_sba\_proxy Class Reference

Degeneralize a [spot::tgba](#) on the fly, producing an SBA.

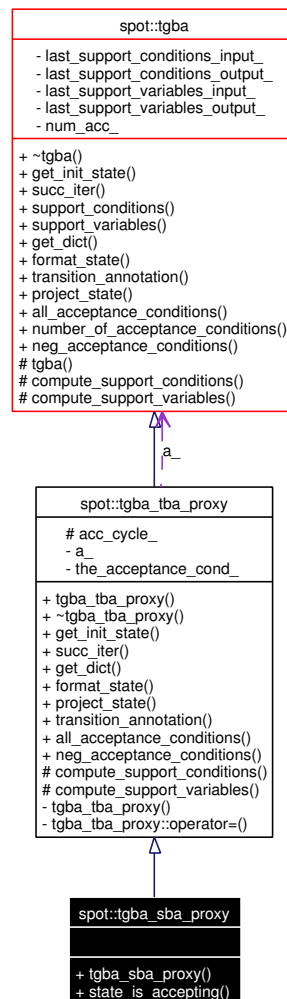
```
#include <tgba/tgbatba.hh>
```

Inheritance diagram for spot::tgba\_sba\_proxy:



Collaboration diagram for spot::tgba\_sba\_proxy:





## Public Types

- typedef std::list< bdd > [cycle\\_list](#)

## Public Member Functions

- [tgba\\_sba\\_proxy](#) (const [tgba](#) \*a)
- bool [state\\_is\\_accepting](#) (const [state](#) \*state) const  
*Whether the state is accepting.*
- virtual [state](#) \* [get\\_init\\_state](#) () const  
*Get the initial state of the automaton.*
- virtual [tgba\\_succ\\_iterator](#) \* [succ\\_iter](#) (const [state](#) \*local\_state, const [state](#) \*global\_state=0, const [tgba](#) \*global\_automaton=0) const  
*Get an iterator over the successors of local\_state.*
- virtual [bdd\\_dict](#) \* [get\\_dict](#) () const

*Get the dictionary associated to the automaton.*

- virtual `std::string format_state` (const `state *state`) const  
*Format the state as a string for printing.*
- virtual `state * project_state` (const `state *s`, const `tgba *t`) const  
*Project a state on an automaton.*
- virtual `std::string transition_annotation` (const `tgba_succ_iterator *t`) const  
*Return a possible annotation for the transition pointed to by the iterator.*
- virtual `bdd all_acceptance_conditions` () const  
*Return the set of all acceptance conditions used by this automaton.*
- virtual `bdd neg_acceptance_conditions` () const  
*Return the conjunction of all negated acceptance variables.*
- `bdd support_conditions` (const `state *state`) const  
*Get a formula that must hold whatever successor is taken.*
- `bdd support_variables` (const `state *state`) const  
*Get the conjunctions of variables tested by the outgoing transitions of state.*
- virtual `int number_of_acceptance_conditions` () const  
*The number of acceptance conditions.*

### Protected Member Functions

- virtual `bdd compute_support_conditions` (const `state *state`) const  
*Do the actual computation of `tgba::support_conditions()`.*
- virtual `bdd compute_support_variables` (const `state *state`) const  
*Do the actual computation of `tgba::support_variables()`.*

### Protected Attributes

- `cycle_list acc_cycle_`

#### 11.105.1 Detailed Description

Degeneralize a `spot::tgba` on the fly, producing an SBA.

This class acts as a proxy in front of a `spot::tgba`, that should be degeneralized on the fly.

This is similar to `tgba_tba_proxy`, except that automata produced with this algorithms can also be seen as State-based Büchi Automata (SBA). See `tgba_sba_proxy::state_is_accepting()`. (An SBA is a TBA, and a TBA is a TGBA.)

This extra property has a small cost in size: if the input automaton uses  $N$  acceptance conditions, the output automaton can have at most  $\max(N,1)+1$  times more states and transitions. (This is only  $\max(N,1)$  for `tgba_tba_proxy`.)

## 11.105.2 Member Typedef Documentation

**11.105.2.1** `typedef std::list<bdd> spot::tgba_tba_proxy::cycle_list` [inherited]

## 11.105.3 Constructor & Destructor Documentation

**11.105.3.1** `spot::tgba_sba_proxy::tgba_sba_proxy (const tgba * a)`

## 11.105.4 Member Function Documentation

**11.105.4.1** `virtual bdd spot::tgba_tba_proxy::all_acceptance_conditions () const` [virtual, inherited]

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::tgba](#).

**11.105.4.2** `virtual bdd spot::tgba_tba_proxy::compute_support_conditions (const state * state) const` [protected, virtual, inherited]

Do the actual computation of [tgba::support\\_conditions\(\)](#).

Implements [spot::tgba](#).

**11.105.4.3** `virtual bdd spot::tgba_tba_proxy::compute_support_variables (const state * state) const` [protected, virtual, inherited]

Do the actual computation of [tgba::support\\_variables\(\)](#).

Implements [spot::tgba](#).

**11.105.4.4** `virtual std::string spot::tgba_tba_proxy::format_state (const state * state) const` [virtual, inherited]

Format the state as a string for printing.

This formatting is the responsibility of the automata who owns the state.

Implements [spot::tgba](#).

**11.105.4.5** `virtual bdd_dict* spot::tgba_tba_proxy::get_dict () const` [virtual, inherited]

Get the dictionary associated to the automaton.

State are represented as BDDs. The dictionary allows to map BDD variables back to formulae, and vice versa. This is useful when dealing with several automata (which may use the same BDD variable for different formula), or simply when printing.

Implements [spot::tgba](#).

**11.105.4.6** `virtual state* spot::tgba_tba_proxy::get_init_state () const [virtual, inherited]`

Get the initial state of the automaton.

The state has been allocated with `new`. It is the responsibility of the caller to delete it when no longer needed.

Implements [spot::tgba](#).

**11.105.4.7** `virtual bdd spot::tgba_tba_proxy::neg_acceptance_conditions () const [virtual, inherited]`

Return the conjunction of all negated acceptance variables.

For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a]&!Acc[b]&!Acc[c]`.

This is useful when making products: each operand's condition set should be augmented with the [neg\\_acceptance\\_conditions\(\)](#) of the other operand.

Implements [spot::tgba](#).

**11.105.4.8** `virtual int spot::tgba::number_of_acceptance_conditions () const [virtual, inherited]`

The number of acceptance conditions.

**11.105.4.9** `virtual state* spot::tgba_tba_proxy::project_state (const state * s, const tgba * t) const [virtual, inherited]`

Project a state on an automaton.

This converts `s`, into that corresponding [spot::state](#) for `t`. This is useful when you have the state of a product, and want restrict this state to a specific automata occurring in the product.

It goes without saying that `s` and `t` should be compatible (i.e., `s` is a state of `t`).

#### Returns:

0 if the projection fails (`s` is unrelated to `t`), or a new `state*` (the projected state) that must be deleted by the caller.

Reimplemented from [spot::tgba](#).

**11.105.4.10** `bool spot::tgba_sba_proxy::state_is_accepting (const state * state) const`

Whether the state is accepting.

A particularity of a `spot::tgba_sba_proxy` automaton is that when a state has an outgoing accepting arc, all its outgoing arcs are accepting. The state itself can therefore be considered accepting. This is useful in algorithms working on degeneralized automata with state acceptance conditions.

**11.105.4.11** `virtual tgba_succ_iterator* spot::tgba_tba_proxy::succ_iter (const state * local_state, const state * global_state = 0, const tgba * global_automaton = 0) const [virtual, inherited]`

Get an iterator over the successors of `local_state`.

The iterator has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

During synchornized products, additional informations are passed about the entire product and its state. Recall that products can be nested, forming a tree of `spot::tgba` where most values are computed on demand. `global_automaton` designate the root `spot::tgba`, and `global_state` its state. This two objects can be used by `succ_iter()` to restrict the set of successors to compute.

#### Parameters:

***local\_state*** The state whose successors are to be explored. This pointer is not adopted in any way by `succ_iter`, and it is still the caller's responsibility to delete it when appropriate (this can be done during the lifetime of the iterator).

***global\_state*** In a product, the state of the global product automaton. Otherwise, 0. Like *locale\_state*, `global_state` is not adopted by `succ_iter`.

***global\_automaton*** In a product, the global product automaton. Otherwise, 0.

Implements `spot::tgba`.

#### 11.105.4.12 bdd spot::tgba::support\_conditions (const state \* state) const [inherited]

Get a formula that must hold whatever successor is taken.

#### Returns:

A formula which must be verified for all successors of *state*.

This can be as simple as `bddtrue`, or more completely the disjunction of the condition of all successors. This is used as an hint by `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_conditions()`, this function is just a wrapper that will cache the last return value for efficiency.

#### 11.105.4.13 bdd spot::tgba::support\_variables (const state \* state) const [inherited]

Get the conjunctions of variables tested by the outgoing transitions of *state*.

All variables tested by outgoing transitions must be returned. This is mandatory.

This is used as an hint by some `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_variables()`, this function is just a wrapper that will cache the last return value for efficiency.

#### 11.105.4.14 virtual std::string spot::tgba\_tba\_proxy::transition\_annotation (const tgba\_succ\_iterator \* t) const [virtual, inherited]

Return a possible annotation for the transition pointed to by the iterator.

Implementing this function is optional; the default annotation it the empty string.

#### Parameters:

***t*** a non-done `tgba_succ_iterator` for this automata

Reimplemented from `spot::tgba`.

### 11.105.5 Member Data Documentation

#### 11.105.5.1 [cycle\\_list](#) [spot::tgba\\_tba\\_proxy::acc\\_cycle\\_](#) [protected, inherited]

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

- [tgba/tgbatba.hh](#)

## 11.106 spot::tgba\_statistics Struct Reference

```
#include <tgbalgorithms/stats.hh>
```

### Public Attributes

- unsigned [transitions](#)
- unsigned [states](#)

### 11.106.1 Member Data Documentation

#### 11.106.1.1 unsigned [spot::tgba\\_statistics::states](#)

#### 11.106.1.2 unsigned [spot::tgba\\_statistics::transitions](#)

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

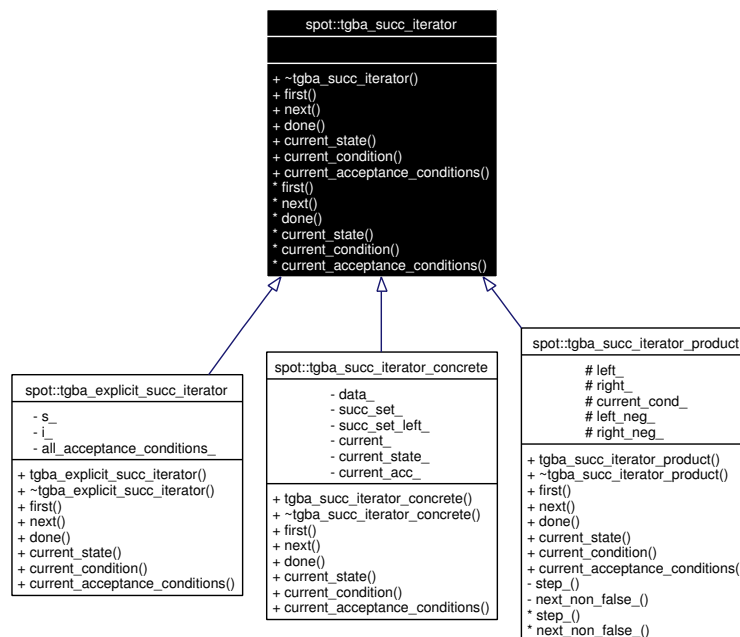
- [tgbalgorithms/stats.hh](#)

## 11.107 spot::tgba\_succ\_iterator Class Reference

Iterate over the successors of a state.

```
#include <tgba/succiter.hh>
```

Inheritance diagram for `spot::tgba_succ_iterator`:



## Public Member Functions

- virtual [~tgba\\_succ\\_iterator](#) ()

## Iteration

- virtual void [first](#) ()=0  
*Position the iterator on the first successor (if any).*
- virtual void [next](#) ()=0  
*Jump to the next successor (if any).*
- virtual bool [done](#) () const =0  
*Check whether the iteration is finished.*

## Inspection

- virtual [state](#) \* [current\\_state](#) () const =0  
*Get the state of the current successor.*
- virtual bdd [current\\_condition](#) () const =0  
*Get the condition on the transition leading to this successor.*
- virtual bdd [current\\_acceptance\\_conditions](#) () const =0  
*Get the acceptance conditions on the transition leading to this successor.*

### 11.107.1 Detailed Description

Iterate over the successors of a state.

This class provides the basic functionalities required to iterate over the successors of a state, as well as querying transition labels. Because transitions are never explicitly encoded, labels (conditions and acceptance conditions) can only be queried while iterating over the successors.

### 11.107.2 Constructor & Destructor Documentation

**11.107.2.1** `virtual spot::tgba_succ_iterator::~tgba_succ_iterator() [inline, virtual]`

### 11.107.3 Member Function Documentation

**11.107.3.1** `virtual bdd spot::tgba_succ_iterator::current_acceptance_conditions() const [pure virtual]`

Get the acceptance conditions on the transition leading to this successor.

Implemented in `spot::tgba_succ_iterator_concrete`, `spot::tgba_explicit_succ_iterator`, and `spot::tgba_succ_iterator_product`.

**11.107.3.2** `virtual bdd spot::tgba_succ_iterator::current_condition() const [pure virtual]`

Get the condition on the transition leading to this successor.

This is a boolean function of atomic propositions.

Implemented in `spot::tgba_succ_iterator_concrete`, `spot::tgba_explicit_succ_iterator`, and `spot::tgba_succ_iterator_product`.

**11.107.3.3** `virtual state* spot::tgba_succ_iterator::current_state() const [pure virtual]`

Get the state of the current successor.

Note that the same state may occur at different points in the iteration. These actually correspond to the same destination. It just means there were several transitions, with different conditions, leading to the same state.

Implemented in `spot::tgba_succ_iterator_concrete`, `spot::tgba_explicit_succ_iterator`, and `spot::tgba_succ_iterator_product`.

**11.107.3.4** `virtual bool spot::tgba_succ_iterator::done() const [pure virtual]`

Check whether the iteration is finished.

This function should be called after any call to `first()` or `next()` and before any enquiry about the current state.

The usual way to do this is with a `for` loop.

```
for (s->first(); !s->done(); s->next())
    ...
```

Implemented in `spot::tgba_succ_iterator_concrete`, `spot::tgba_explicit_succ_iterator`, and `spot::tgba_succ_iterator_product`.



**11.107.3.5 virtual void spot::tgba\_succ\_iterator::first () [pure virtual]**

Position the iterator on the first successor (if any).

This method can be called several times to make multiple passes over successors.

**Warning:**

One should always call `done()` to ensure there is a successor, even after `first()`. A common trap is to assume that there is at least one successor: this is wrong.

Implemented in `spot::tgba_succ_iterator_concrete`, `spot::tgba_explicit_succ_iterator`, and `spot::tgba_succ_iterator_product`.

**11.107.3.6 virtual void spot::tgba\_succ\_iterator::next () [pure virtual]**

Jump to the next successor (if any).

**Warning:**

Again, one should always call `done()` to ensure there is a successor.

Implemented in `spot::tgba_succ_iterator_concrete`, `spot::tgba_explicit_succ_iterator`, and `spot::tgba_succ_iterator_product`.

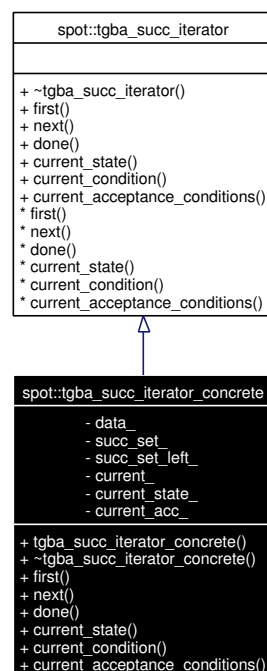
The documentation for this class was generated from the following file:

- [tgba/succiter.hh](#)

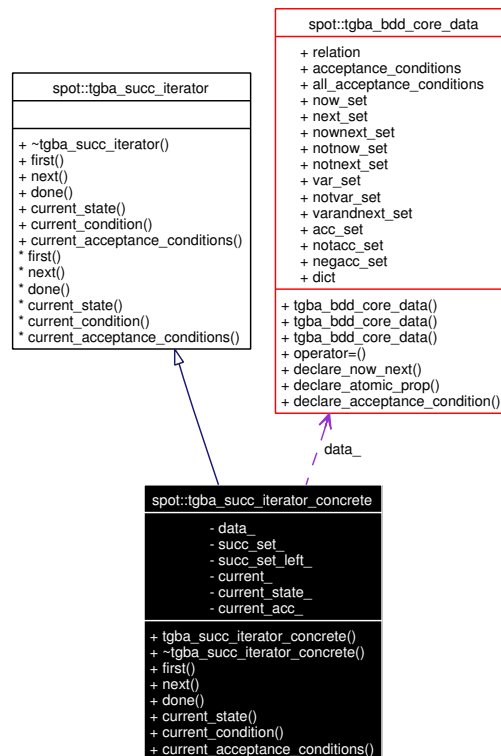
**11.108 spot::tgba\_succ\_iterator\_concrete Class Reference**

```
#include <tgba/succiterconcrete.hh>
```

Inheritance diagram for `spot::tgba_succ_iterator_concrete`:



Collaboration diagram for spot::tgba\_succ\_iterator\_concrete:



## Public Member Functions

- [tgba\\_succ\\_iterator\\_concrete](#) (const [tgba\\_bdd\\_core\\_data](#) &d, bdd successors)  
Build a `spot::tgba_succ_iterator_concrete`.
- virtual [~tgba\\_succ\\_iterator\\_concrete](#) ()
- void [first](#) ()  
Position the iterator on the first successor (if any).
- void [next](#) ()  
Jump to the next successor (if any).
- bool [done](#) () const  
Check whether the iteration is finished.
- [state\\_bdd](#) \* [current\\_state](#) () const  
Get the state of the current successor.
- bdd [current\\_condition](#) () const  
Get the condition on the transition leading to this successor.
- bdd [current\\_acceptance\\_conditions](#) () const  
Get the acceptance conditions on the transition leading to this successor.

### Private Attributes

- const [tgba\\_bdd\\_core\\_data](#) & [data\\_](#)  
*Core data of the automaton.*
- bdd [succ\\_set\\_](#)  
*The set of successors.*
- bdd [succ\\_set\\_left\\_](#)  
*Unexplored successors (including current\_).*
- bdd [current\\_](#)  
*Current successor, as a conjunction of atomic proposition and Next variables.*
- bdd [current\\_state\\_](#)  
*Current successor, as a conjunction of Now variables.*
- bdd [current\\_acc\\_](#)  
*Accepting conditions for the current transition.*

#### 11.108.1 Detailed Description

A concrete iterator over successors of a TGBA state.

#### 11.108.2 Constructor & Destructor Documentation

##### 11.108.2.1 spot::tgba\_succ\_iterator\_concrete::tgba\_succ\_iterator\_concrete (const [tgba\\_bdd\\_core\\_data](#) & *d*, bdd *successors*)

Build a spot::tgba\_succ\_iterator\_concrete.

##### Parameters:

- successors* The set of successors with ingoing conditions and acceptance conditions, represented as a BDD. The job of this iterator will be to enumerate the satisfactions of that BDD and split them into destination states and conditions, and compute acceptance conditions.
- d* The core data of the automata. These contains sets of variables useful to split a BDD, and compute acceptance conditions.

##### 11.108.2.2 virtual spot::tgba\_succ\_iterator\_concrete::~tgba\_succ\_iterator\_concrete () [virtual]

#### 11.108.3 Member Function Documentation

##### 11.108.3.1 bdd spot::tgba\_succ\_iterator\_concrete::current\_acceptance\_conditions () const [virtual]

Get the acceptance conditions on the transition leading to this successor.

Implements [spot::tgba\\_succ\\_iterator](#).

**11.108.3.2** `bdd spot::tgba_succ_iterator_concrete::current_condition () const` [virtual]

Get the condition on the transition leading to this successor.

This is a boolean function of atomic propositions.

Implements [spot::tgba\\_succ\\_iterator](#).

**11.108.3.3** `state_bdd* spot::tgba_succ_iterator_concrete::current_state () const` [virtual]

Get the state of the current successor.

Note that the same state may occur at different points in the iteration. These actually correspond to the same destination. It just means there were several transitions, with different conditions, leading to the same state.

Implements [spot::tgba\\_succ\\_iterator](#).

**11.108.3.4** `bool spot::tgba_succ_iterator_concrete::done () const` [virtual]

Check whether the iteration is finished.

This function should be called after any call to `first()` or `next()` and before any enquiry about the current state.

The usual way to do this is with a `for` loop.

```
for (s->first(); !s->done(); s->next())
    ...
```

Implements [spot::tgba\\_succ\\_iterator](#).

**11.108.3.5** `void spot::tgba_succ_iterator_concrete::first ()` [virtual]

Position the iterator on the first successor (if any).

This method can be called several times to make multiple passes over successors.

**Warning:**

One should always call `done()` to ensure there is a successor, even after `first()`. A common trap is to assume that there is at least one successor: this is wrong.

Implements [spot::tgba\\_succ\\_iterator](#).

**11.108.3.6** `void spot::tgba_succ_iterator_concrete::next ()` [virtual]

Jump to the next successor (if any).

**Warning:**

Again, one should always call `done()` to ensure there is a successor.

Implements [spot::tgba\\_succ\\_iterator](#).

**11.108.4** Member Data Documentation**11.108.4.1** `bdd spot::tgba_succ_iterator_concrete::current_` [private]

Current successor, as a conjunction of atomic proposition and Next variables.

**11.108.4.2** bdd [spot::tgba\\_succ\\_iterator\\_concrete::current\\_acc\\_](#) [private]

Accepting conditions for the current transition.

**11.108.4.3** bdd [spot::tgba\\_succ\\_iterator\\_concrete::current\\_state\\_](#) [private]

Current successor, as a conjunction of Now variables.

**11.108.4.4** const [tgba\\_bdd\\_core\\_data&](#) [spot::tgba\\_succ\\_iterator\\_concrete::data\\_](#) [private]

Core data of the automaton.

**11.108.4.5** bdd [spot::tgba\\_succ\\_iterator\\_concrete::succ\\_set\\_](#) [private]

The set of successors.

**11.108.4.6** bdd [spot::tgba\\_succ\\_iterator\\_concrete::succ\\_set\\_left\\_](#) [private]

Unexplored successors (including current\_).

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

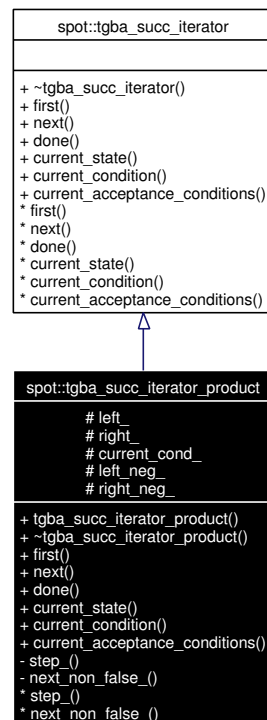
- [tgba/succiterconcrete.hh](#)

## 11.109 spot::tgba\_succ\_iterator\_product Class Reference

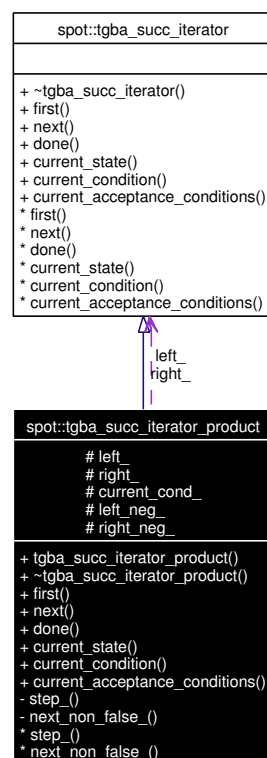
Iterate over the successors of a product computed on the fly.

```
#include <tgba/tgbaproduct.hh>
```

Inheritance diagram for spot::tgba\_succ\_iterator\_product:



Collaboration diagram for `spot::tgba_succ_iterator_product`:



## Public Member Functions

- [tgba\\_succ\\_iterator\\_product](#) ([tgba\\_succ\\_iterator](#) \*left, [tgba\\_succ\\_iterator](#) \*right, bdd left\_neg, bdd right\_neg)
- virtual [~tgba\\_succ\\_iterator\\_product](#) ()
- void [first](#) ()  
*Position the iterator on the first successor (if any).*
- void [next](#) ()  
*Jump to the next successor (if any).*
- bool [done](#) () const  
*Check whether the iteration is finished.*
- [state\\_product](#) \* [current\\_state](#) () const  
*Get the state of the current successor.*
- bdd [current\\_condition](#) () const  
*Get the condition on the transition leading to this successor.*
- bdd [current\\_acceptance\\_conditions](#) () const  
*Get the acceptance conditions on the transition leading to this successor.*

## Protected Attributes

- [tgba\\_succ\\_iterator](#) \* left\_
- [tgba\\_succ\\_iterator](#) \* right\_
- bdd [current\\_cond\\_](#)
- bdd [left\\_neg\\_](#)
- bdd [right\\_neg\\_](#)

## Private Member Functions

- void [step\\_](#) ()  
*Internal routines to advance to the next successor.*
- void [next\\_non\\_false\\_](#) ()

## Friends

- class [tgba\\_product](#)

### 11.109.1 Detailed Description

Iterate over the successors of a product computed on the fly.

## 11.109.2 Constructor & Destructor Documentation

**11.109.2.1** `spot::tgba_succ_iterator_product::tgba_succ_iterator_product (tgba_succ_iterator * left, tgba_succ_iterator * right, bdd left_neg, bdd right_neg)`

**11.109.2.2** `virtual spot::tgba_succ_iterator_product::~tgba_succ_iterator_product ()` [virtual]

## 11.109.3 Member Function Documentation

**11.109.3.1** `bdd spot::tgba_succ_iterator_product::current_acceptance_conditions () const` [virtual]

Get the acceptance conditions on the transition leading to this successor.

Implements [spot::tgba\\_succ\\_iterator](#).

**11.109.3.2** `bdd spot::tgba_succ_iterator_product::current_condition () const` [virtual]

Get the condition on the transition leading to this successor.

This is a boolean function of atomic propositions.

Implements [spot::tgba\\_succ\\_iterator](#).

**11.109.3.3** `state_product* spot::tgba_succ_iterator_product::current_state () const` [virtual]

Get the state of the current successor.

Note that the same state may occur at different points in the iteration. These actually correspond to the same destination. It just means there were several transitions, with different conditions, leading to the same state.

Implements [spot::tgba\\_succ\\_iterator](#).

**11.109.3.4** `bool spot::tgba_succ_iterator_product::done () const` [virtual]

Check whether the iteration is finished.

This function should be called after any call to [first\(\)](#) or [next\(\)](#) and before any enquiry about the current state.

The usual way to do this is with a `for` loop.

```
for (s->first(); !s->done(); s->next())
    ...
```

Implements [spot::tgba\\_succ\\_iterator](#).

**11.109.3.5** `void spot::tgba_succ_iterator_product::first ()` [virtual]

Position the iterator on the first successor (if any).

This method can be called several times to make multiple passes over successors.



**Warning:**

One should always call `done()` to ensure there is a successor, even after `first()`. A common trap is to assume that there is at least one successor: this is wrong.

Implements `spot::tgba_succ_iterator`.

**11.109.3.6** `void spot::tgba_succ_iterator_product::next()` [virtual]

Jump to the next successor (if any).

**Warning:**

Again, one should always call `done()` to ensure there is a successor.

Implements `spot::tgba_succ_iterator`.

**11.109.3.7** `void spot::tgba_succ_iterator_product::next_non_false_()` [private]**11.109.3.8** `void spot::tgba_succ_iterator_product::step_()` [private]

Internal routines to advance to the next successor.

**11.109.4 Friends And Related Function Documentation****11.109.4.1** friend class `tgba_product` [friend]**11.109.5 Member Data Documentation****11.109.5.1** `bdd spot::tgba_succ_iterator_product::current_cond_` [protected]**11.109.5.2** `tgba_succ_iterator* spot::tgba_succ_iterator_product::left_` [protected]**11.109.5.3** `bdd spot::tgba_succ_iterator_product::left_neg_` [protected]**11.109.5.4** `tgba_succ_iterator* spot::tgba_succ_iterator_product::right_` [protected]**11.109.5.5** `bdd spot::tgba_succ_iterator_product::right_neg_` [protected]

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

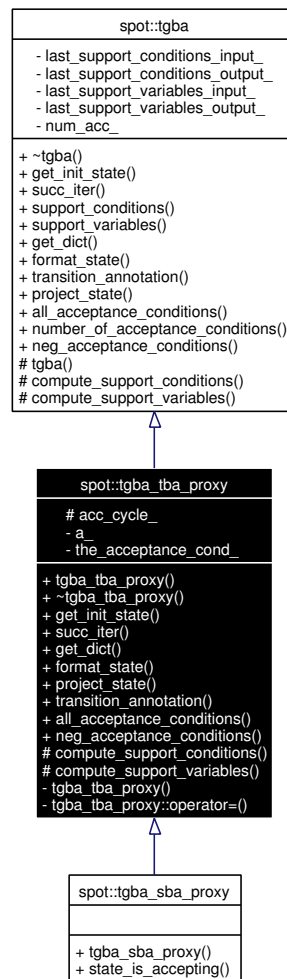
- `tgba/tgbaproduct.hh`

**11.110 `spot::tgba_tba_proxy` Class Reference**

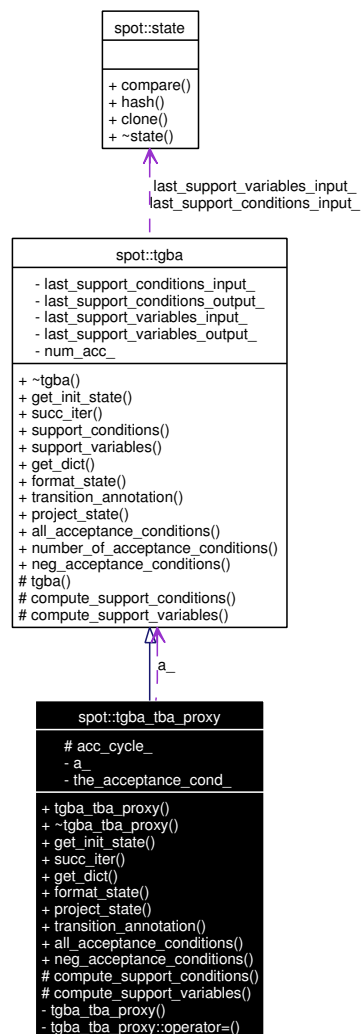
Degeneralize a `spot::tgba` on the fly, producing a TBA.

```
#include <tgba/tgbatba.hh>
```

Inheritance diagram for `spot::tgba_tba_proxy`:



Collaboration diagram for spot::tgba\_tba\_proxy:



## Public Types

- typedef std::list< bdd > [cycle\\_list](#)

## Public Member Functions

- [tgba\\_tba\\_proxy](#) (const [tgba](#) \*a)
- virtual [~tgba\\_tba\\_proxy](#) ()
- virtual [state](#) \* [get\\_init\\_state](#) () const  
*Get the initial state of the automaton.*
- virtual [tgba\\_succ\\_iterator](#) \* [succ\\_iter](#) (const [state](#) \*local\_state, const [state](#) \*global\_state=0, const [tgba](#) \*global\_automaton=0) const  
*Get an iterator over the successors of local\_state.*
- virtual [bdd\\_dict](#) \* [get\\_dict](#) () const  
*Get the dictionary associated to the automaton.*

- virtual std::string [format\\_state](#) (const [state](#) \*state) const  
*Format the state as a string for printing.*
- virtual [state](#) \* [project\\_state](#) (const [state](#) \*s, const [tgba](#) \*t) const  
*Project a state on an automaton.*
- virtual std::string [transition\\_annotation](#) (const [tgba\\_succ\\_iterator](#) \*t) const  
*Return a possible annotation for the transition pointed to by the iterator.*
- virtual bdd [all\\_acceptance\\_conditions](#) () const  
*Return the set of all acceptance conditions used by this automaton.*
- virtual bdd [neg\\_acceptance\\_conditions](#) () const  
*Return the conjunction of all negated acceptance variables.*
- bdd [support\\_conditions](#) (const [state](#) \*state) const  
*Get a formula that must hold whatever successor is taken.*
- bdd [support\\_variables](#) (const [state](#) \*state) const  
*Get the conjunctions of variables tested by the outgoing transitions of state.*
- virtual int [number\\_of\\_acceptance\\_conditions](#) () const  
*The number of acceptance conditions.*

### Protected Member Functions

- virtual bdd [compute\\_support\\_conditions](#) (const [state](#) \*state) const  
*Do the actual computation of [tgba::support\\_conditions\(\)](#).*
- virtual bdd [compute\\_support\\_variables](#) (const [state](#) \*state) const  
*Do the actual computation of [tgba::support\\_variables\(\)](#).*

### Protected Attributes

- [cycle\\_list](#) [acc\\_cycle\\_](#)

### Private Member Functions

- [tgba\\_tba\\_proxy](#) (const [tgba\\_tba\\_proxy](#) &)
- [tgba\\_tba\\_proxy](#) & [tgba\\_tba\\_proxy::operator=](#) (const [tgba\\_tba\\_proxy](#) &)

### Private Attributes

- const [tgba](#) \* [a\\_](#)
- bdd [the\\_acceptance\\_cond\\_](#)

### 11.110.1 Detailed Description

Degeneralize a `spot::tgba` on the fly, producing a TBA.

This class acts as a proxy in front of a `spot::tgba`, that should be degeneralized on the fly. The result is still a `spot::tgba`, but it will always have exactly one acceptance condition so it could be called TBA (without the G).

The degeneralization is done by synchronizing the input automaton with a "counter" automaton such as the one shown in "On-the-fly Verification of Linear Temporal Logic" (Jean-Michel Couvreur, FME99).

If the input automaton uses  $N$  acceptance conditions, the output automaton can have at most  $\max(N,1)$  times more states and transitions.

See also:

[`tgba\_sba\_proxy`](#)

### 11.110.2 Member Typedef Documentation

**11.110.2.1** `typedef std::list<bdd> spot::tgba_tba_proxy::cycle_list`

### 11.110.3 Constructor & Destructor Documentation

**11.110.3.1** `spot::tgba_tba_proxy::tgba_tba_proxy (const tgba * a)`

**11.110.3.2** `virtual spot::tgba_tba_proxy::~tgba\_tba\_proxy ()` [virtual]

**11.110.3.3** `spot::tgba_tba_proxy::tgba_tba_proxy (const tgba\_tba\_proxy &)` [private]

### 11.110.4 Member Function Documentation

**11.110.4.1** `virtual bdd spot::tgba_tba_proxy::all_acceptance_conditions () const` [virtual]

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implements [`spot::tgba`](#).

**11.110.4.2** `virtual bdd spot::tgba_tba_proxy::compute_support_conditions (const state * state) const` [protected, virtual]

Do the actual computation of [`tgba::support\_conditions\(\)`](#).

Implements [`spot::tgba`](#).

**11.110.4.3** `virtual bdd spot::tgba_tba_proxy::compute_support_variables (const state * state) const` [protected, virtual]

Do the actual computation of [`tgba::support\_variables\(\)`](#).

Implements [`spot::tgba`](#).

**11.110.4.4** `virtual std::string spot::tgba_tba_proxy::format_state (const state * state) const [virtual]`

Format the state as a string for printing.

This formatting is the responsibility of the automata who owns the state.

Implements [spot::tgba](#).

**11.110.4.5** `virtual bdd_dict* spot::tgba_tba_proxy::get_dict () const [virtual]`

Get the dictionary associated to the automaton.

State are represented as BDDs. The dictionary allows to map BDD variables back to formulae, and vice versa. This is useful when dealing with several automata (which may use the same BDD variable for different formula), or simply when printing.

Implements [spot::tgba](#).

**11.110.4.6** `virtual state* spot::tgba_tba_proxy::get_init_state () const [virtual]`

Get the initial state of the automaton.

The state has been allocated with new. It is the responsibility of the caller to delete it when no longer needed.

Implements [spot::tgba](#).

**11.110.4.7** `virtual bdd spot::tgba_tba_proxy::neg_acceptance_conditions () const [virtual]`

Return the conjunction of all negated acceptance variables.

For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a]&!Acc[b]&!Acc[c]`.

This is useful when making products: each operand's condition set should be augmented with the [neg\\_acceptance\\_conditions\(\)](#) of the other operand.

Implements [spot::tgba](#).

**11.110.4.8** `virtual int spot::tgba::number_of_acceptance_conditions () const [virtual, inherited]`

The number of acceptance conditions.

**11.110.4.9** `virtual state* spot::tgba_tba_proxy::project_state (const state * s, const tgba * t) const [virtual]`

Project a state on an automaton.

This converts *s*, into that corresponding [spot::state](#) for *t*. This is useful when you have the state of a product, and want restrict this state to a specific automata occurring in the product.

It goes without saying that *s* and *t* should be compatible (i.e., *s* is a state of *t*).

#### Returns:

0 if the projection fails (*s* is unrelated to *t*), or a new `state*` (the projected state) that must be deleted by the caller.

Reimplemented from [spot::tgba](#).

**11.110.4.10** virtual [tgba\\_succ\\_iterator](#)\* [spot::tgba\\_tba\\_proxy::succ\\_iter](#) (const [state](#) \* *local\_state*, const [state](#) \* *global\_state* = 0, const [tgba](#) \* *global\_automaton* = 0) const [virtual]

Get an iterator over the successors of *local\_state*.

The iterator has been allocated with new. It is the responsibility of the caller to delete it when no longer needed.

During synchornized products, additional informations are passed about the entire product and its state. Recall that products can be nested, forming a tree of [spot::tgba](#) where most values are computed on demand. *global\_automaton* designate the root [spot::tgba](#), and *global\_state* its state. This two objects can be used by [succ\\_iter\(\)](#) to restrict the set of successors to compute.

#### Parameters:

*local\_state* The state whose successors are to be explored. This pointer is not adopted in any way by [succ\\_iter](#), and it is still the caller's responsibility to delete it when appropriate (this can be done during the lifetime of the iterator).

*global\_state* In a product, the state of the global product automaton. Otherwise, 0. Like *locale\_state*, *global\_state* is not adopted by [succ\\_iter](#).

*global\_automaton* In a product, the global product automaton. Otherwise, 0.

Implements [spot::tgba](#).

**11.110.4.11** bdd [spot::tgba::support\\_conditions](#) (const [state](#) \* *state*) const [inherited]

Get a formula that must hold whatever successor is taken.

#### Returns:

A formula which must be verified for all successors of *state*.

This can be as simple as `bddtrue`, or more completely the disjunction of the condition of all successors. This is used as an hint by [succ\\_iter\(\)](#) to reduce the number of successor to compute in a product.

Sub classes should implement [compute\\_support\\_conditions\(\)](#), this function is just a wrapper that will cache the last return value for efficiency.

**11.110.4.12** bdd [spot::tgba::support\\_variables](#) (const [state](#) \* *state*) const [inherited]

Get the conjunctions of variables tested by the outgoing transitions of *state*.

All variables tested by outgoing transitions must be returned. This is mandatory.

This is used as an hint by some [succ\\_iter\(\)](#) to reduce the number of successor to compute in a product.

Sub classes should implement [compute\\_support\\_variables\(\)](#), this function is just a wrapper that will cache the last return value for efficiency.

**11.110.4.13** [tgba\\_tba\\_proxy](#)& [spot::tgba\\_tba\\_proxy::tgba\\_tba\\_proxy::operator=](#) (const [tgba\\_tba\\_proxy](#) &) [private]

**11.110.4.14** virtual std::string [spot::tgba\\_tba\\_proxy::transition\\_annotation](#) (const [tgba\\_succ\\_iterator](#) \* *t*) const [virtual]

Return a possible annotation for the transition pointed to by the iterator.

Implementing this function is optional; the default annotation it the empty string.

**Parameters:**

*t* a non-done [tgba\\_succ\\_iterator](#) for this automata

Reimplemented from [spot::tgba](#).

**11.110.5 Member Data Documentation**

**11.110.5.1** `const tgba* spot::tgba_tba_proxy::a_` [private]

**11.110.5.2** `cycle_list spot::tgba_tba_proxy::acc_cycle_` [protected]

**11.110.5.3** `bdd spot::tgba_tba_proxy::the_acceptance_cond_` [private]

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

- [tgba/tgbatba.hh](#)

**11.111 `spot::time_info` Struct Reference**

A structure to record elapsed time in clock ticks.

```
#include <misc/timer.hh>
```

**Public Member Functions**

- [time\\_info](#) ()

**Public Attributes**

- `clock_t` [utime](#)
- `clock_t` [stime](#)

**11.111.1 Detailed Description**

A structure to record elapsed time in clock ticks.

**11.111.2 Constructor & Destructor Documentation**

**11.111.2.1** `spot::time_info::time_info ()` [inline]

**11.111.3 Member Data Documentation**

**11.111.3.1** `clock_t spot::time_info::stime`

**11.111.3.2** `clock_t spot::time_info::utime`

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

- [misc/timer.hh](#)

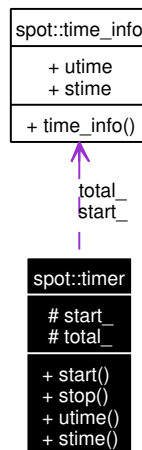


## 11.112 spot::timer Class Reference

A timekeeper that accumulate interval of time.

```
#include <misc/timer.hh>
```

Collaboration diagram for spot::timer:



### Public Member Functions

- void [start](#) ()  
*Start a time interval.*
- void [stop](#) ()  
*Stop a time interval and update the sum of all intervals.*
- clock\_t [utime](#) () const  
*Return the user time of all accumulated interval.*
- clock\_t [stime](#) () const  
*Return the system time of all accumulated interval.*

### Protected Attributes

- [time\\_info](#) [start\\_](#)
- [time\\_info](#) [total\\_](#)

#### 11.112.1 Detailed Description

A timekeeper that accumulate interval of time.

### 11.112.2 Member Function Documentation

#### 11.112.2.1 `void spot::timer::start () [inline]`

Start a time interval.

#### 11.112.2.2 `clock_t spot::timer::stime () const [inline]`

Return the system time of all accumulated interval.

Any time interval that has been `start()`ed but not `stop()`ed will not be accounted for.

#### 11.112.2.3 `void spot::timer::stop () [inline]`

Stop a time interval and update the sum of all intervals.

#### 11.112.2.4 `clock_t spot::timer::utime () const [inline]`

Return the user time of all accumulated interval.

Any time interval that has been `start()`ed but not `stop()`ed will not be accounted for.

### 11.112.3 Member Data Documentation

#### 11.112.3.1 `time_info spot::timer::start_ [protected]`

#### 11.112.3.2 `time_info spot::timer::total_ [protected]`

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

- `misc/timer.hh`

## 11.113 `spot::timer_map` Class Reference

A map of timer, where each timer has a name.

```
#include <misc/timer.hh>
```

### Public Member Functions

- `void start (const std::string &name)`  
*Start a timer with name name.*
- `void stop (const std::string &name)`  
*Stop timer name.*
- `void cancel (const std::string &name)`  
*Cancel timer name.*
- `const spot::timer & timer (const std::string &name) const`  
*Return the timer name.*

- `spot::timer & timer` (const std::string &name)  
*Return the timer name.*
- bool `empty` () const  
*Whether there is no timer in the map.*
- std::ostream & `print` (std::ostream &os) const  
*Format information about all timers in a table.*

### Protected Types

- typedef std::pair< `spot::timer`, int > `item_type`
- typedef std::map< std::string, `item_type` > `tm_type`

### Protected Attributes

- `tm_type tm`

#### 11.113.1 Detailed Description

A map of timer, where each timer has a name.

Timer\_map also keeps track of the number of measures each timer has performed.

#### 11.113.2 Member Typedef Documentation

**11.113.2.1** typedef std::pair<`spot::timer`, int> `spot::timer_map::item_type` [protected]

**11.113.2.2** typedef std::map<std::string, `item_type`> `spot::timer_map::tm_type` [protected]

#### 11.113.3 Member Function Documentation

**11.113.3.1** void `spot::timer_map::cancel` (const std::string & *name*) [inline]

Cancel timer *name*.

The timer must have been previously started with `start()`.

This cancel only the current measure. (Previous measures recorded by the timer are preserved.) When a timer that has not done any measure is canceled, it is removed from the map.

**11.113.3.2** bool `spot::timer_map::empty` () const [inline]

Whether there is no timer in the map.

If `empty()` return true, then either no timer where ever started, or all started timers were canceled without completing any measure.

**11.113.3.3** `std::ostream& spot::timer_map::print (std::ostream & os) const`

Format information about all timers in a table.

**11.113.3.4** `void spot::timer_map::start (const std::string & name) [inline]`

Start a timer with name *name*.

The timer is created if it did not exist already. Once started, a timer should be either [stop\(\)](#)ed or [cancel\(\)](#)ed.

**11.113.3.5** `void spot::timer_map::stop (const std::string & name) [inline]`

Stop timer *name*.

The timer must have been previously started with [start\(\)](#).

**11.113.3.6** `spot::timer& spot::timer_map::timer (const std::string & name) [inline]`

Return the timer *name*.

**11.113.3.7** `const spot::timer& spot::timer_map::timer (const std::string & name) const [inline]`

Return the timer *name*.

**11.113.4** Member Data Documentation**11.113.4.1** `tm_type spot::timer_map::tm [protected]`

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

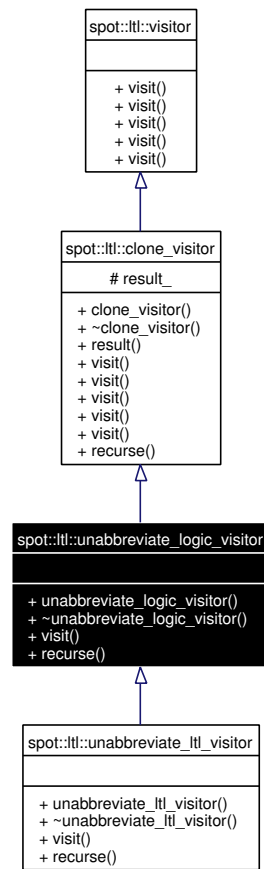
- [misc/timer.hh](#)

**11.114** `spot::ltl::unabbreviate_logic_visitor` Class Reference

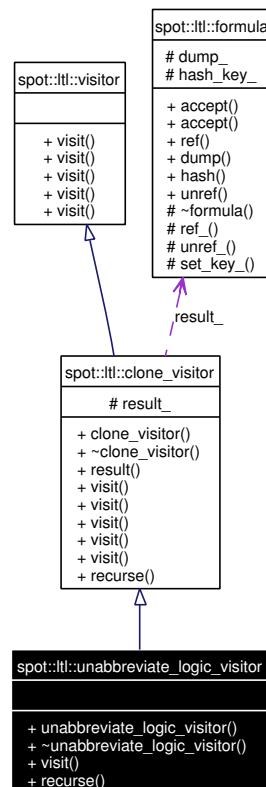
Clone and rewrite a formula to remove most of the abbreviated logical operators.

```
#include <ltlvisit/ltlunabbrev.hh>
```

Inheritance diagram for `spot::ltl::unabbreviate_logic_visitor`:



Collaboration diagram for `spot::ltl::unabbreviate_logic_visitor`:



## Public Member Functions

- `unabbreviate_logic_visitor ()`
- `virtual ~unabbreviate_logic_visitor ()`
- `void visit (binop *bo)`
- `virtual formula * recurse (formula *f)`
- `formula * result () const`
- `void visit (atomic_prop *ap)`
- `void visit (unop *uo)`
- `void visit (multop *mo)`
- `void visit (constant *c)`

## Protected Attributes

- `formula * result_`

## Private Types

- `typedef clone_visitor super`

### 11.114.1 Detailed Description

Clone and rewrite a formula to remove most of the abbreviated logical operators.

This will rewrite binary operators such as [binop::Implies](#), [binop::Equals](#), and [binop::Xor](#), using only [unop::Not](#), [multop::Or](#), and [multop::And](#).

This visitor is public, because it's convenient to derive from it and override some of its methods. But if you just want the functionality, consider using [spot::ltl::unabbreviate\\_logic](#) instead.

### 11.114.2 Member Typedef Documentation

**11.114.2.1** typedef [clone\\_visitor](#) [spot::ltl::unabbreviate\\_logic\\_visitor::super](#) [private]

Reimplemented in [spot::ltl::unabbreviate\\_ltl\\_visitor](#).

### 11.114.3 Constructor & Destructor Documentation

**11.114.3.1** [spot::ltl::unabbreviate\\_logic\\_visitor::unabbreviate\\_logic\\_visitor](#) ()

**11.114.3.2** virtual [spot::ltl::unabbreviate\\_logic\\_visitor::~~unabbreviate\\_logic\\_visitor](#) ()  
[virtual]

### 11.114.4 Member Function Documentation

**11.114.4.1** virtual [formula\\*](#) [spot::ltl::unabbreviate\\_logic\\_visitor::recurse](#) ([formula](#) \* *f*)  
[virtual]

Reimplemented from [spot::ltl::clone\\_visitor](#).

Reimplemented in [spot::ltl::unabbreviate\\_ltl\\_visitor](#).

**11.114.4.2** [formula\\*](#) [spot::ltl::clone\\_visitor::result](#) () const [inherited]

**11.114.4.3** void [spot::ltl::clone\\_visitor::visit](#) ([constant](#) \* *c*) [virtual, inherited]

Implements [spot::ltl::visitor](#).

**11.114.4.4** void [spot::ltl::clone\\_visitor::visit](#) ([multop](#) \* *mo*) [virtual, inherited]

Implements [spot::ltl::visitor](#).

**11.114.4.5** void [spot::ltl::clone\\_visitor::visit](#) ([unop](#) \* *uo*) [virtual, inherited]

Implements [spot::ltl::visitor](#).

Reimplemented in [spot::ltl::unabbreviate\\_ltl\\_visitor](#).

**11.114.4.6** void [spot::ltl::clone\\_visitor::visit](#) ([atomic\\_prop](#) \* *ap*) [virtual, inherited]

Implements [spot::ltl::visitor](#).

**11.114.4.7** void [spot::ltl::unabbreviate\\_logic\\_visitor::visit](#) ([binop](#) \* *bo*) [virtual]

Reimplemented from [spot::ltl::clone\\_visitor](#).

### 11.114.5 Member Data Documentation

#### 11.114.5.1 formula\* spot::ltl::clone\_visitor::result\_ [protected, inherited]

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

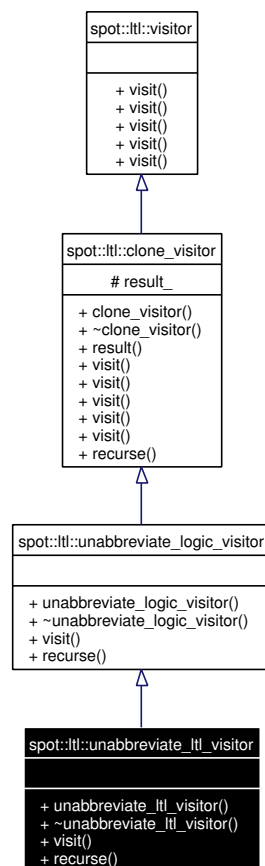
- [ltlvisit/unabbrev.hh](#)

## 11.115 spot::ltl::unabbreviate\_ltl\_visitor Class Reference

Clone and rewrite a formula to remove most of the abbreviated LTL and logical operators.

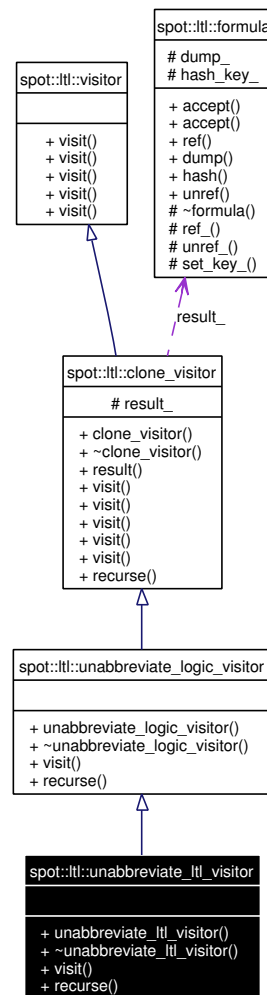
```
#include <ltlvisit/tunabbrev.hh>
```

Inheritance diagram for spot::ltl::unabbreviate\_ltl\_visitor:



Collaboration diagram for spot::ltl::unabbreviate\_ltl\_visitor:





## Public Member Functions

- `unabbreviate_ltl_visitor ()`
- `virtual ~unabbreviate_ltl_visitor ()`
- `void visit (unop *uo)`
- `formula * recurse (formula *f)`
- `void visit (binop *bo)`
- `void visit (atomic_prop *ap)`
- `void visit (multop *mo)`
- `void visit (constant *c)`
- `formula * result () const`

## Protected Attributes

- `formula * result_`

## Private Types

- typedef `unabbreviate_logic_visitor` `super`

### 11.115.1 Detailed Description

Clone and rewrite a formula to remove most of the abbreviated LTL and logical operators.

The rewriting performed on logical operator is the same as the one done by `spot::ltl::unabbreviate_logic_visitor`.

This will also rewrite unary operators such as `unop::F`, and `unop::G`, using only `binop::U`, and `binop::R`.

This visitor is public, because it's convenient to derive from it and override some of its methods. But if you just want the functionality, consider using `spot::ltl::unabbreviate_ltl` instead.

### 11.115.2 Member Typedef Documentation

**11.115.2.1** typedef `unabbreviate_logic_visitor` `spot::ltl::unabbreviate_ltl_visitor::super` [private]

Reimplemented from `spot::ltl::unabbreviate_logic_visitor`.

### 11.115.3 Constructor & Destructor Documentation

**11.115.3.1** `spot::ltl::unabbreviate_ltl_visitor::unabbreviate_ltl_visitor ()`

**11.115.3.2** virtual `spot::ltl::unabbreviate_ltl_visitor::~~unabbreviate_ltl_visitor ()` [virtual]

### 11.115.4 Member Function Documentation

**11.115.4.1** `formula*` `spot::ltl::unabbreviate_ltl_visitor::recurse (formula *f)` [virtual]

Reimplemented from `spot::ltl::unabbreviate_logic_visitor`.

**11.115.4.2** `formula*` `spot::ltl::clone_visitor::result () const` [inherited]

**11.115.4.3** void `spot::ltl::clone_visitor::visit (constant *c)` [virtual, inherited]

Implements `spot::ltl::visitor`.

**11.115.4.4** void `spot::ltl::clone_visitor::visit (multop *mo)` [virtual, inherited]

Implements `spot::ltl::visitor`.

**11.115.4.5** void `spot::ltl::clone_visitor::visit (atomic_prop *ap)` [virtual, inherited]

Implements `spot::ltl::visitor`.

**11.115.4.6** void spot::ltl::unabbreviate\_logic\_visitor::visit (binop \* bo) [virtual, inherited]

Reimplemented from [spot::ltl::clone\\_visitor](#).

**11.115.4.7** void spot::ltl::unabbreviate\_ltl\_visitor::visit (unop \* uo) [virtual]

Reimplemented from [spot::ltl::clone\\_visitor](#).

### 11.115.5 Member Data Documentation

**11.115.5.1** formula\* spot::ltl::clone\_visitor::result\_ [protected, inherited]

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

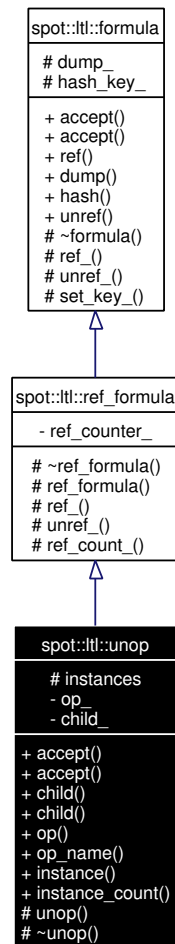
- [ltlvisit/tunabbrev.hh](#)

## 11.116 spot::ltl::unop Class Reference

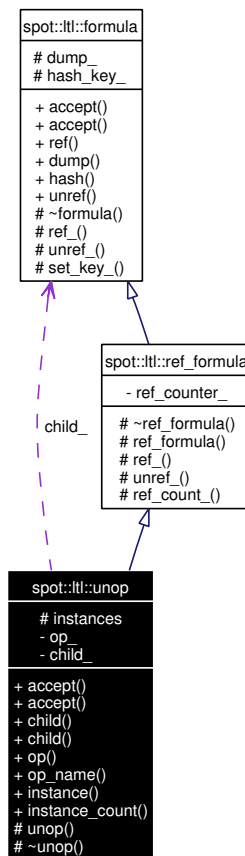
Unary operators.

```
#include <ltlast/unop.hh>
```

Inheritance diagram for spot::ltl::unop:



Collaboration diagram for `spot::ltl::unop`:



## Public Types

- enum `type` { `Not`, `X`, `F`, `G` }

## Public Member Functions

- virtual void `accept` (`visitor` &`v`)  
*Entry point for `vspot::ltl::visitor` instances.*
- virtual void `accept` (`const_visitor` &`v`) `const`  
*Entry point for `vspot::ltl::const_visitor` instances.*
- `const formula` \* `child` () `const`  
*Get the sole operand of this operator.*
- `formula` \* `child` ()  
*Get the sole operand of this operator.*
- `type` `op` () `const`  
*Get the type of this operator.*
- `const char` \* `op_name` () `const`

*Get the type of this operator, as a string.*

- `formula * ref ()`  
*clone this node*
- `const std::string & dump () const`  
*Return a canonic representation of the formula.*
- `const size_t hash () const`  
*Return a hash\_key for the formula.*

### Static Public Member Functions

- `static unop * instance (type op, formula *child)`
- `static unsigned instance_count ()`  
*Number of instantiated unary operators. For debugging.*
- `static void unref (formula *f)`  
*release this node*

### Protected Types

- `typedef std::pair< type, formula * > pair`
- `typedef std::map< pair, formula * > map`

### Protected Member Functions

- `unop (type op, formula *child)`
- `virtual ~unop ()`
- `void ref_ ()`  
*increment reference counter if any*
- `bool unref_ ()`  
*decrement reference counter if any, return true when the instance must be deleted (usually when the counter hits 0).*
- `unsigned ref_count_ ()`  
*Number of references to this formula.*
- `void set_key_ ()`  
*Compute key\_ from dump\_.*

**Protected Attributes**

- std::string [dump\\_](#)  
*The canonic representation of the formula.*
- size\_t [hash\\_key\\_](#)  
*The hash key of this formula.*

**Static Protected Attributes**

- static [map instances](#)

**Private Attributes**

- [type op\\_](#)
- [formula \\* child\\_](#)

**11.116.1 Detailed Description**

Unary operators.

**11.116.2 Member Typedef Documentation**

**11.116.2.1** typedef std::map<[pair](#), [formula\\*](#)> [spot::ltl::unop::map](#) [protected]

**11.116.2.2** typedef std::pair<[type](#), [formula\\*](#)> [spot::ltl::unop::pair](#) [protected]

**11.116.3 Member Enumeration Documentation**

**11.116.3.1** enum [spot::ltl::unop::type](#)

Enumeration values:

*Not*

*X*

*F*

*G*

**11.116.4 Constructor & Destructor Documentation**

**11.116.4.1** [spot::ltl::unop::unop](#) ([type op](#), [formula \\* child](#)) [protected]

**11.116.4.2** virtual [spot::ltl::unop::~~unop](#) () [protected, virtual]

### 11.116.5 Member Function Documentation

#### 11.116.5.1 virtual void spot::ltl::unop::accept (const\_visitor & v) const [virtual]

Entry point for vspot::ltl::const\_visitor instances.

Implements [spot::ltl::formula](#).

#### 11.116.5.2 virtual void spot::ltl::unop::accept (visitor & v) [virtual]

Entry point for vspot::ltl::visitor instances.

Implements [spot::ltl::formula](#).

#### 11.116.5.3 formula\* spot::ltl::unop::child ()

Get the sole operand of this operator.

#### 11.116.5.4 const formula\* spot::ltl::unop::child () const

Get the sole operand of this operator.

#### 11.116.5.5 const std::string& spot::ltl::formula::dump () const [inherited]

Return a canonic representation of the formula.

#### 11.116.5.6 const size\_t spot::ltl::formula::hash () const [inline, inherited]

Return a hash\_key for the formula.

#### 11.116.5.7 static unop\* spot::ltl::unop::instance (type op, formula \* child) [static]

Build an unary operator with operation *op* and child *child*.

#### 11.116.5.8 static unsigned spot::ltl::unop::instance\_count () [static]

Number of instantiated unary operators. For debugging.

#### 11.116.5.9 type spot::ltl::unop::op () const

Get the type of this operator.

#### 11.116.5.10 const char\* spot::ltl::unop::op\_name () const

Get the type of this operator, as a string.

#### 11.116.5.11 formula\* spot::ltl::formula::ref () [inherited]

clone this node

This increments the reference counter of this node (if one is used). You should almost never use this method directly as it doesn't touch the children. If you want to clone a whole formula, use [spot::ltl::clone\(\)](#) instead.



**11.116.5.12** `void spot::ltl::ref_formula::ref_()` [protected, virtual, inherited]

increment reference counter if any

Reimplemented from [spot::ltl::formula](#).

**11.116.5.13** `unsigned spot::ltl::ref_formula::ref_count_()` [protected, inherited]

Number of references to this formula.

**11.116.5.14** `void spot::ltl::formula::set_key_()` [protected, inherited]

Compute `key_` from `dump_`.

Should be called once in each object, after `dump_` has been set.

**11.116.5.15** `static void spot::ltl::formula::unref(formula *f)` [static, inherited]

release this node

This decrements the reference counter of this node (if one is used) and can free the object. You should almost never use this method directly as it doesn't touch the children. If you want to release a whole formula, use [spot::ltl::destroy\(\)](#) instead.

**11.116.5.16** `bool spot::ltl::ref_formula::unref_()` [protected, virtual, inherited]

decrement reference counter if any, return true when the instance must be deleted (usually when the counter hits 0).

Reimplemented from [spot::ltl::formula](#).

**11.116.6** Member Data Documentation**11.116.6.1** `formula* spot::ltl::unop::child_` [private]**11.116.6.2** `std::string spot::ltl::formula::dump_` [protected, inherited]

The canonic representation of the formula.

**11.116.6.3** `size_t spot::ltl::formula::hash_key_` [protected, inherited]

The hash key of this formula.

Initialized by [set\\_key\\_\(\)](#).

**11.116.6.4** `map spot::ltl::unop::instances` [static, protected]**11.116.6.5** `type spot::ltl::unop::op_` [private]

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

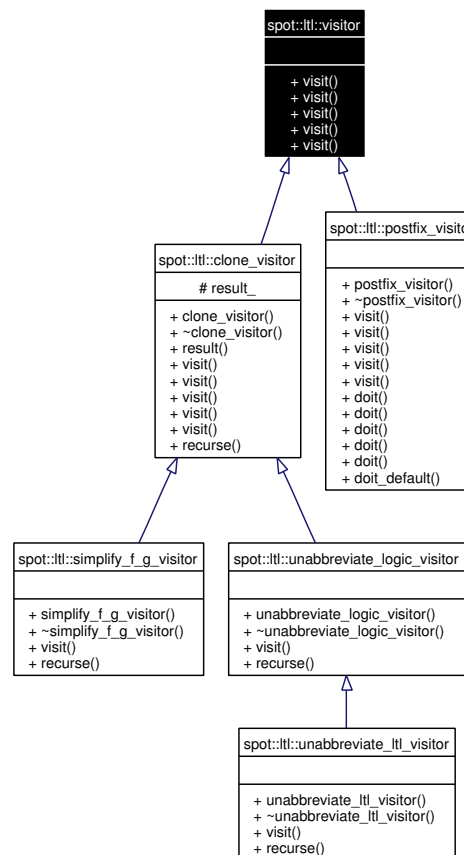
- [ltlast/unop.hh](#)

## 11.117 spot::ltl::visitor Struct Reference

Formula visitor that can modify the formula.

```
#include <ltlast/visitor.hh>
```

Inheritance diagram for spot::ltl::visitor:



### Public Member Functions

- virtual void [visit](#) ([atomic\\_prop](#) \*node)=0
- virtual void [visit](#) ([constant](#) \*node)=0
- virtual void [visit](#) ([binop](#) \*node)=0
- virtual void [visit](#) ([unop](#) \*node)=0
- virtual void [visit](#) ([multop](#) \*node)=0

#### 11.117.1 Detailed Description

Formula visitor that can modify the formula.

Writing visitors is the preferred way to traverse a formula, since it doesn't involve any cast.

If you do not need to modify the visited formula, inherit from [spot::ltl::const\\_visitor](#) instead.

### 11.117.2 Member Function Documentation

**11.117.2.1 virtual void spot::ltl::visitor::visit (multop \* node)** [pure virtual]

Implemented in [spot::ltl::clone\\_visitor](#), and [spot::ltl::postfix\\_visitor](#).

**11.117.2.2 virtual void spot::ltl::visitor::visit (unop \* node)** [pure virtual]

Implemented in [spot::ltl::clone\\_visitor](#), [spot::ltl::postfix\\_visitor](#), and [spot::ltl::unabbreviate\\_ltl\\_visitor](#).

**11.117.2.3 virtual void spot::ltl::visitor::visit (binop \* node)** [pure virtual]

Implemented in [spot::ltl::clone\\_visitor](#), [spot::ltl::unabbreviate\\_logic\\_visitor](#), [spot::ltl::postfix\\_visitor](#), and [spot::ltl::simplify\\_f\\_g\\_visitor](#).

**11.117.2.4 virtual void spot::ltl::visitor::visit (constant \* node)** [pure virtual]

Implemented in [spot::ltl::clone\\_visitor](#), and [spot::ltl::postfix\\_visitor](#).

**11.117.2.5 virtual void spot::ltl::visitor::visit (atomic\_prop \* node)** [pure virtual]

Implemented in [spot::ltl::clone\\_visitor](#), and [spot::ltl::postfix\\_visitor](#).

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

- [ltlast/visitor.hh](#)

## 11.118 spot::weight Class Reference

Manage for a given automaton a vector of counter indexed by its acceptance condition.

```
#include <tgbaaalgos/weight.hh>
```

### Public Member Functions

- [weight](#) (const bdd &neg\_all\_cond)
- [weight](#) & [operator+=](#) (const bdd &acc)  
*Increment by one the counters of each acceptance condition in acc.*
- [weight](#) & [operator-=](#) (const bdd &acc)  
*Decrement by one the counters of each acceptance condition in acc.*
- bdd [operator-](#) (const [weight](#) &w) const

### Private Types

- typedef std::map< int, int > [weight\\_vector](#)

### Static Private Member Functions

- static void [inc\\_weight\\_handler](#) (char \*varset, int size)
- static void [dec\\_weight\\_handler](#) (char \*varset, int size)

**Private Attributes**

- [weight\\_vector](#) *m*
- bdd [neg\\_all\\_acc](#)

**Static Private Attributes**

- static [weight\\_vector](#) \* *pm*

**Friends**

- `std::ostream & operator<< (std::ostream &os, const weight &w)`

**11.118.1 Detailed Description**

Manage for a given automaton a vector of counter indexed by its acceptance condition.

**11.118.2 Member Typedef Documentation**

**11.118.2.1** `typedef std::map<int, int> spot::weight::weight\_vector [private]`

**11.118.3 Constructor & Destructor Documentation**

**11.118.3.1** `spot::weight::weight (const bdd & neg_all_cond)`

Construct a empty vector (all counters set to zero).

**Parameters:**

*neg\_all\_cond* : negation of all the acceptance conditions of the automaton (the bdd returned by [tgba::neg\\_acceptance\\_conditions\(\)](#)).

**11.118.4 Member Function Documentation**

**11.118.4.1** `static void spot::weight::dec_weight_handler (char * varset, int size) [static, private]`

**11.118.4.2** `static void spot::weight::inc_weight_handler (char * varset, int size) [static, private]`

**11.118.4.3** `weight& spot::weight::operator+= (const bdd & acc)`

Increment by one the counters of each acceptance condition in *acc*.

**11.118.4.4** `bdd spot::weight::operator- (const weight & w) const`

Return the set of each acceptance condition such that its counter is strictly greatest than the corresponding counter in *w*.

**Precondition:**

For each acceptance condition, its counter is greatest or equal to the corresponding counter in *w*.

**11.118.4.5** `weight` & `spot::weight::operator-= (const bdd & acc)`

Decrement by one the counters of each acceptance condition in *acc*.

**11.118.5** Friends And Related Function Documentation**11.118.5.1** `std::ostream& operator<< (std::ostream & os, const weight & w)` [`friend`]**11.118.6** Member Data Documentation**11.118.6.1** `weight_vector spot::weight::m` [`private`]**11.118.6.2** `bdd spot::weight::neg_all_acc` [`private`]**11.118.6.3** `weight_vector* spot::weight::pm` [`static, private`]

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

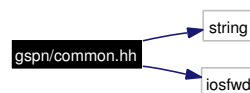
- `tgbaalgos/weight.hh`

**12** spot File Documentation**12.1** `mainpage.dox` File Reference**12.2** `gspn/common.hh` File Reference

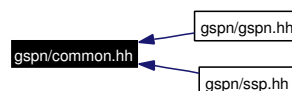
```
#include <string>
```

```
#include <iosfwd>
```

Include dependency graph for `common.hh`:



This graph shows which files directly or indirectly include this file:

**Namespaces**

- namespace `spot`

## Functions

- `std::ostream & operator<< (std::ostream &os, const gspn_exception &e)`

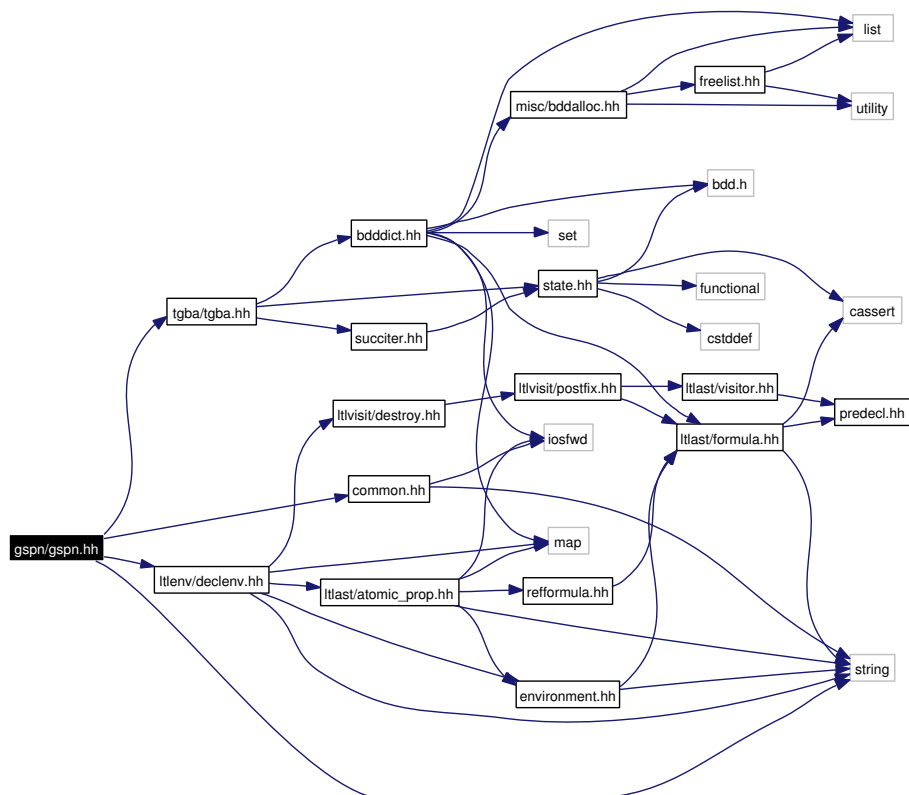
### 12.2.1 Function Documentation

#### 12.2.1.1 `std::ostream& operator<< (std::ostream &os, const gspn_exception &e)`

## 12.3 gspn/gspn.hh File Reference

```
#include <string>
#include "tgba/tgba.hh"
#include "common.hh"
#include "ltlenv/declenv.hh"
```

Include dependency graph for gspn.hh:



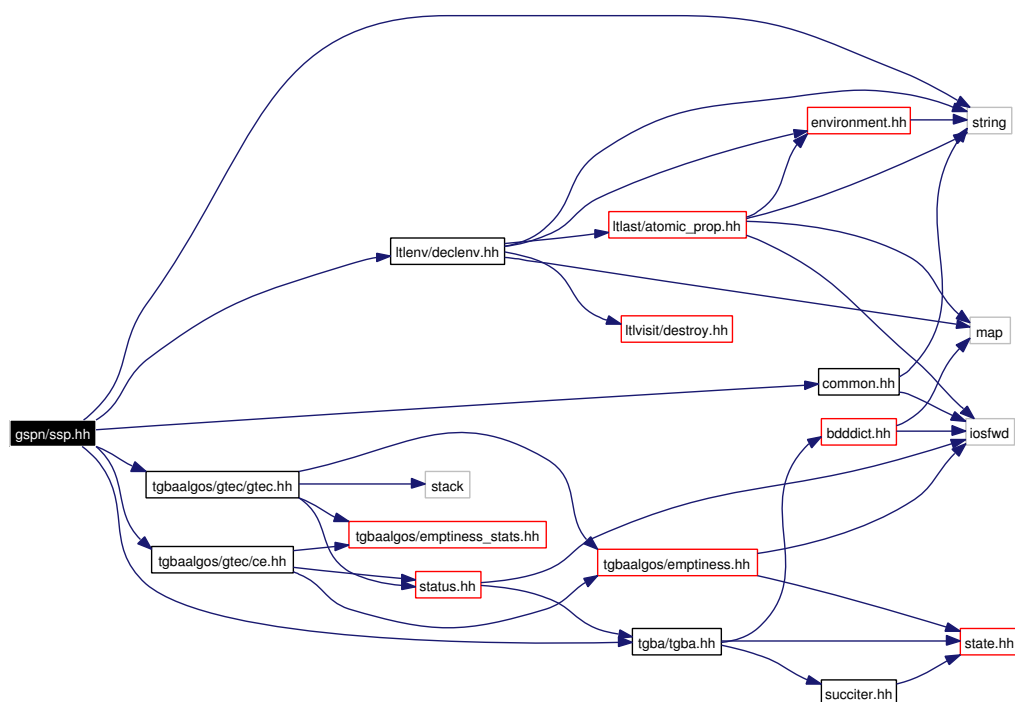
## Namespaces

- namespace `spot`

## 12.4 gspn/ssp.hh File Reference

```
#include <string>
#include "tgba/tgba.hh"
#include "common.hh"
#include "tgbaalgos/gtec/gtec.hh"
#include "tgbaalgos/gtec/ce.hh"
#include "ltenv/declenv.hh"
```

Include dependency graph for ssp.hh:



### Namespaces

- namespace [spot](#)

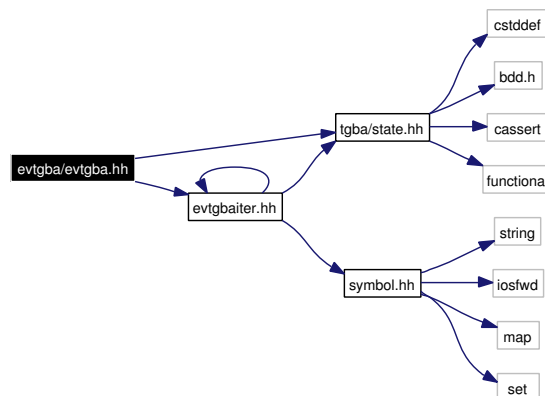
### Functions

- `couvreur99_check * couvreur99_check_ssp_semi` (const tgba \*ssp\_automata)
- `couvreur99_check * couvreur99_check_ssp_shy_semi` (const tgba \*ssp\_automata)
- `couvreur99_check * couvreur99_check_ssp_shy` (const tgba \*ssp\_automata)

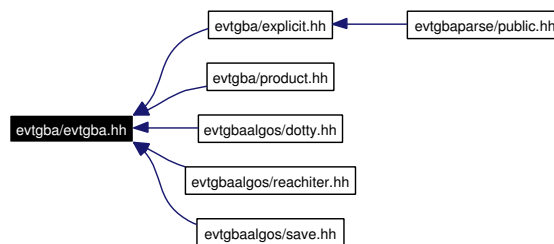
## 12.5 evtgba/evtgba.hh File Reference

```
#include "tgba/state.hh"
#include "evtgbaiter.hh"
```

Include dependency graph for evtgba.hh:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace `spot`

## 12.6 evtgba/evtgbaiter.hh File Reference

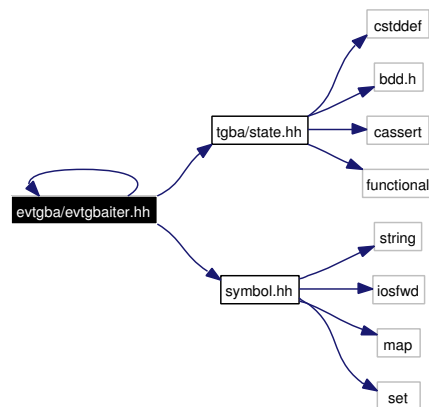
```
#include "tgba/state.hh"
```

```
#include "symbol.hh"
```

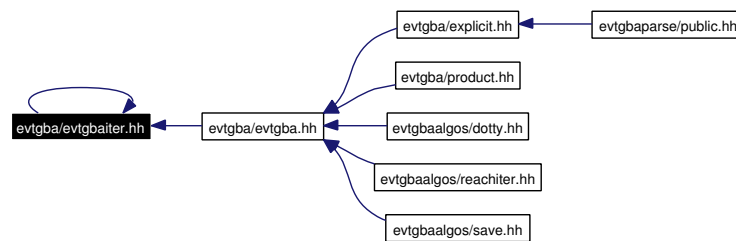
```
#include "evtgbaiter.hh"
```

Include dependency graph for evtgbaiter.hh:





This graph shows which files directly or indirectly include this file:



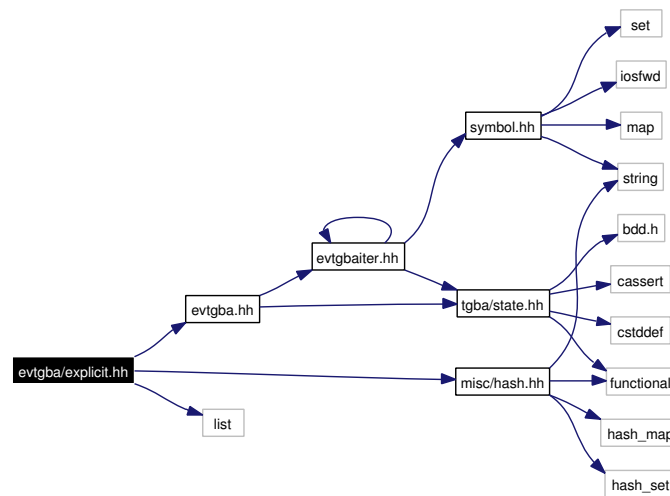
## Namespaces

- namespace `spot`

## 12.7 evtgba/explicit.hh File Reference

```
#include "evtgba.hh"
#include <list>
#include "misc/hash.hh"
```

Include dependency graph for `explicit.hh`:



This graph shows which files directly or indirectly include this file:



## Namespaces

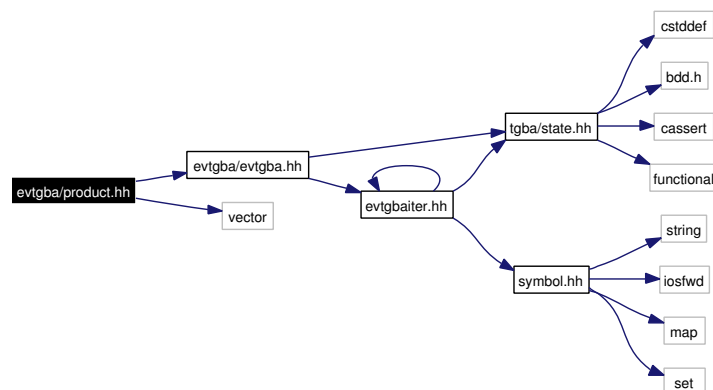
- namespace [spot](#)

## 12.8 evtgba/product.hh File Reference

```
#include "evtgba/evtgba.hh"
```

```
#include <vector>
```

Include dependency graph for product.hh:



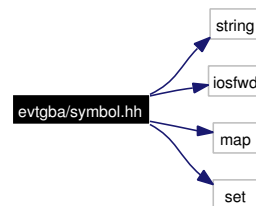
## Namespaces

- namespace [spot](#)

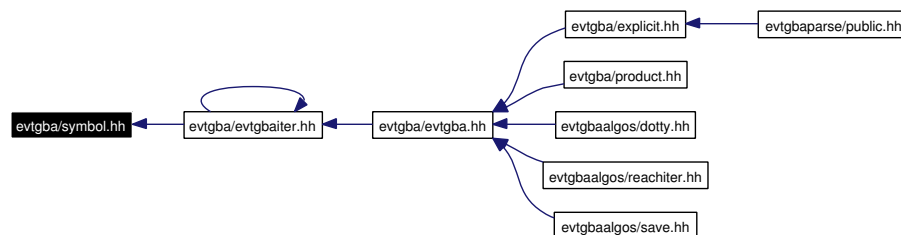
## 12.9 evtgba/symbol.hh File Reference

```
#include <string>
#include <iosfwd>
#include <map>
#include <set>
```

Include dependency graph for symbol.hh:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [spot](#)

### Typedefs

- typedef std::set< const symbol \* > [symbol\\_set](#)
- typedef std::set< rsymbol > [rsymbol\\_set](#)

#### 12.9.1 Typedef Documentation

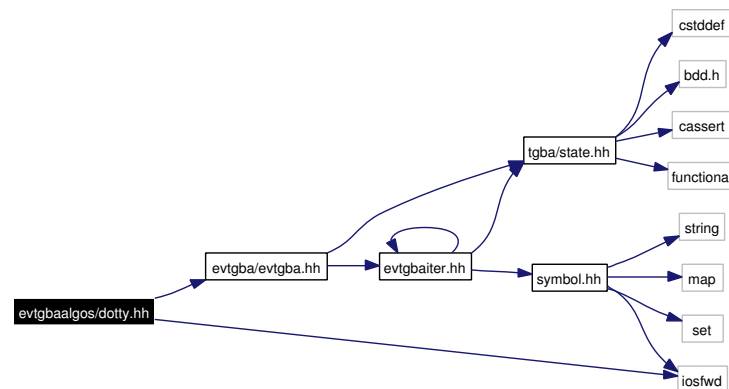
##### 12.9.1.1 typedef std::set<rsymbol> [spot::rsymbol\\_set](#)

##### 12.9.1.2 typedef std::set<const symbol\*> [spot::symbol\\_set](#)

## 12.10 evtgbaalgos/dotty.hh File Reference

```
#include "evtgba/evtgba.hh"
#include <iosfwd>
```

Include dependency graph for dotty.hh:



## Namespaces

- namespace [spot](#)

## Functions

- `std::ostream & dotty\_reachable (std::ostream &os, const evtgba *g)`  
*Print reachable states in dot format.*

### 12.10.1 Function Documentation

#### 12.10.1.1 `std::ostream& dotty_reachable (std::ostream &os, const evtgba *g)`

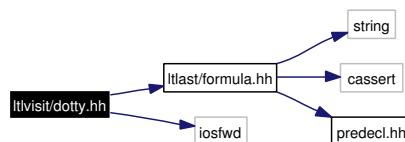
Print reachable states in dot format.

## 12.11 ltlvisit/dotty.hh File Reference

```
#include <ltlast/formula.hh>
```

```
#include <iosfwd>
```

Include dependency graph for dotty.hh:



## Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

## Functions

- `std::ostream & dotty (std::ostream &os, const formula *f)`

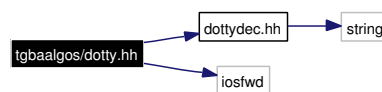
*Write a formula tree using dot's syntax.*

## 12.12 tgbaalgos/dotty.hh File Reference

```
#include "dottydec.hh"
```

```
#include <iosfwd>
```

Include dependency graph for dotty.hh:



## Namespaces

- namespace `spot`

## Functions

- `std::ostream & dotty_reachable (std::ostream &os, const tgba *g, dotty_decorator *dd=dotty_decorator::instance())`

*Print reachable states in dot format.*

## 12.13 evtgbaalgos/reachiter.hh File Reference

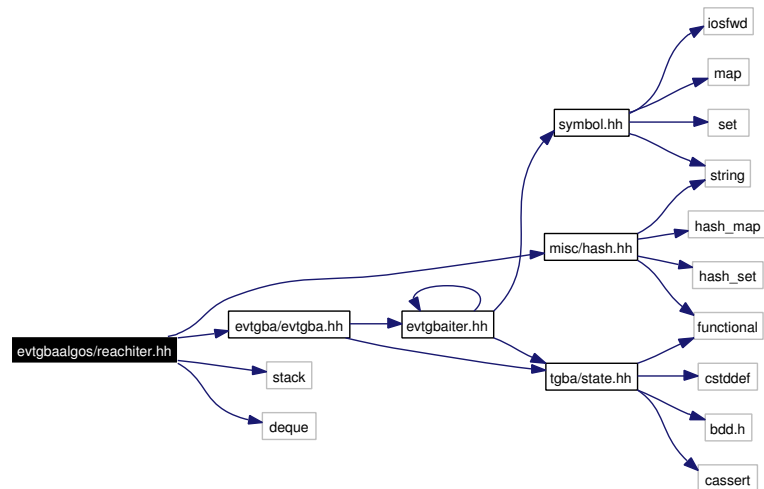
```
#include "misc/hash.hh"
```

```
#include "evtgba/evtgba.hh"
```

```
#include <stack>
```

```
#include <deque>
```

Include dependency graph for reachiter.hh:



## Namespaces

- namespace [spot](#)

## 12.14 tgbaalgos/reachiter.hh File Reference

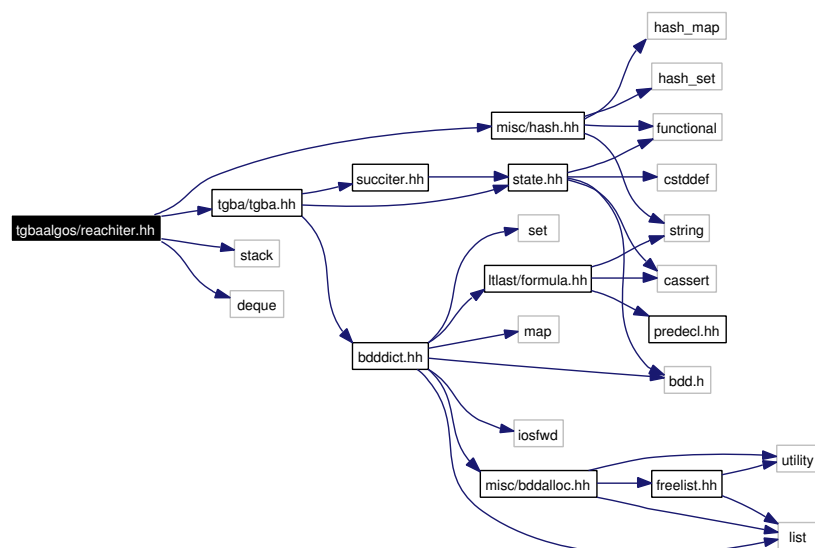
```
#include "misc/hash.hh"
```

```
#include "tgba/tgba.hh"
```

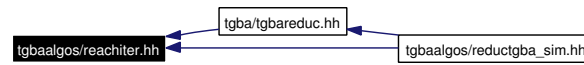
```
#include <stack>
```

```
#include <deque>
```

Include dependency graph for reachiter.hh:



This graph shows which files directly or indirectly include this file:



## Namespaces

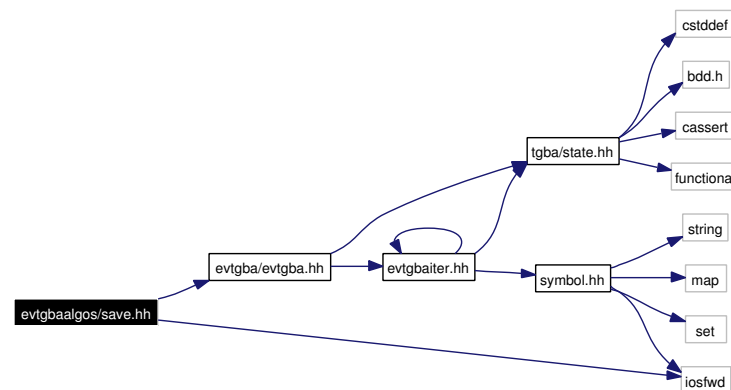
- namespace [spot](#)

## 12.15 evtgbaalgos/save.hh File Reference

```
#include "evtgba/evtgba.hh"
```

```
#include <iosfwd>
```

Include dependency graph for save.hh:



## Namespaces

- namespace [spot](#)

## Functions

- `std::ostream & evtgba\_save\_reachable (std::ostream &os, const evtgba *g)`  
*Save reachable states in text format.*

### 12.15.1 Function Documentation

#### 12.15.1.1 `std::ostream& evtgba_save_reachable (std::ostream &os, const evtgba *g)`

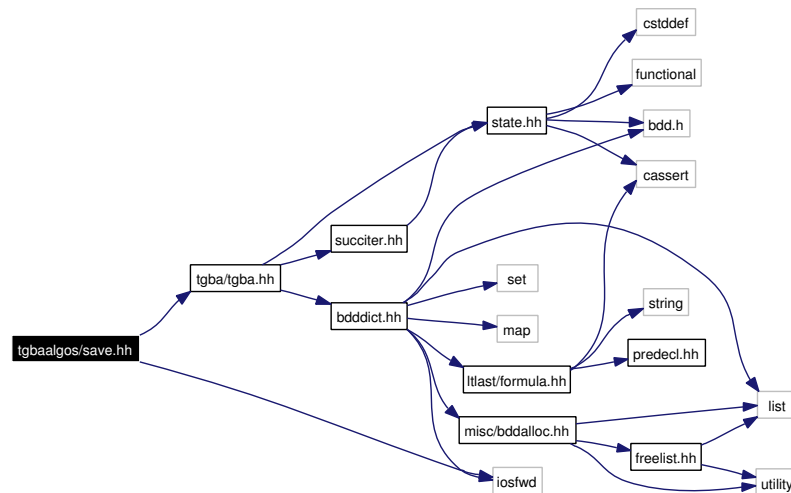
Save reachable states in text format.

## 12.16 tgbaalgos/save.hh File Reference

```
#include "tgba/tgba.hh"
```

```
#include <iosfwd>
```

Include dependency graph for save.hh:



### Namespaces

- namespace [spot](#)

### Functions

- `std::ostream & tgba\_save\_reachable (std::ostream &os, const tgba *g)`  
*Save reachable states in text format.*

## 12.17 evtgbaalgos/tgba2evtgba.hh File Reference

### Namespaces

- namespace [spot](#)

### Functions

- `evtgba_explicit * tgba\_to\_evtgba (const tgba *a)`  
*Convert a tgba into an evtgba.*

### 12.17.1 Function Documentation

#### 12.17.1.1 evtgba\_explicit\* tgba\_to\_evtgba (const tgba \* a)



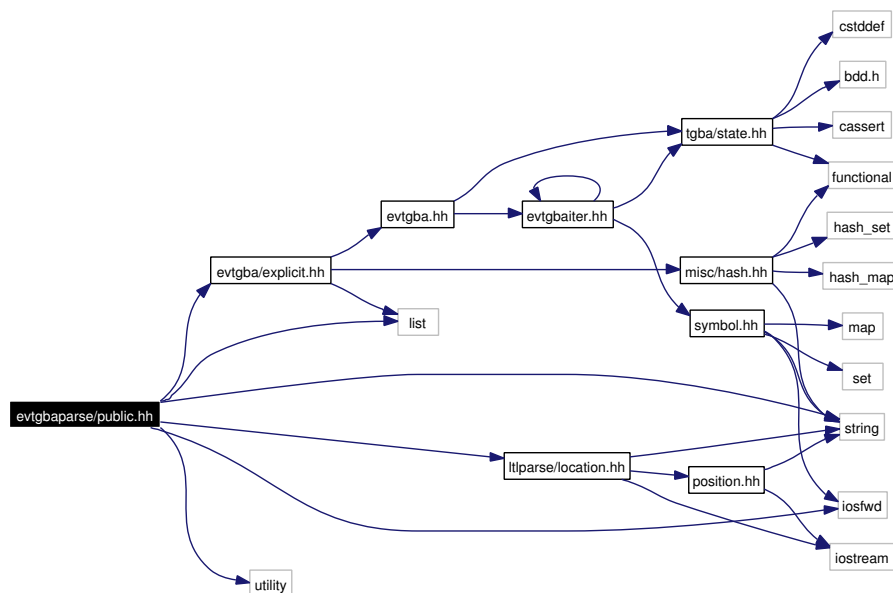
Convert a tgba into an evtgba.

(This cannot be done on-the-fly because the alphabet of a tgba is unknown beforehand.)

## 12.18 evtgba<sub>parse</sub>/public.hh File Reference

```
#include "evtgba/explicit.hh"
#include "ltlparse/location.hh"
#include <string>
#include <list>
#include <utility>
#include <iosfwd>
```

Include dependency graph for public.hh:



### Namespaces

- namespace [spot](#)

### Typedefs

- typedef std::pair< [yy::Location](#), std::string > [evtgba\\_parse\\_error](#)  
A parse diagnostic with its location.
- typedef std::list< [evtgba\\_parse\\_error](#) > [evtgba\\_parse\\_error\\_list](#)  
A list of parser diagnostics, as filled by parse.

## Functions

- `evtgba_explicit * evtgba_parse (const std::string &filename, evtgba_parse_error_list &error_list, bool debug=false)`  
Build a `spot::evtgba_explicit` from a text file.
- `bool format_evtgba_parse_errors (std::ostream &os, evtgba_parse_error_list &error_list)`  
Format diagnostics produced by `spot::evtgba_parse`.

### 12.18.1 Typedef Documentation

#### 12.18.1.1 `typedef std::pair<yy::Location, std::string> spot::evtgba_parse_error`

A parse diagnostic with its location.

#### 12.18.1.2 `typedef std::list<evtgba_parse_error> spot::evtgba_parse_error_list`

A list of parser diagnostics, as filled by parse.

### 12.18.2 Function Documentation

#### 12.18.2.1 `evtgba_explicit* evtgba_parse (const std::string &filename, evtgba_parse_error_list &error_list, bool debug = false)`

Build a `spot::evtgba_explicit` from a text file.

##### Parameters:

*filename* The name of the file to parse.

*error\_list* A list that will be filled with parse errors that occurred during parsing.

*debug* When true, causes the parser to trace its execution.

##### Returns:

A pointer to the `evtgba` built from *filename*, or 0 if the file could not be opened.

Note that the parser usually tries to recover from errors. It can return a non zero value even if it encountered error during the parsing of *filename*. If you want to make sure *filename* was parsed successfully, check *error\_list* for emptiness.

##### Warning:

This function is not reentrant.

#### 12.18.2.2 `bool format_evtgba_parse_errors (std::ostream &os, evtgba_parse_error_list &error_list)`

Format diagnostics produced by `spot::evtgba_parse`.

##### Parameters:

*os* Where diagnostics should be output.

*error\_list* The error list filled by `spot::ltl::parse` while parsing *ltl\_string*.

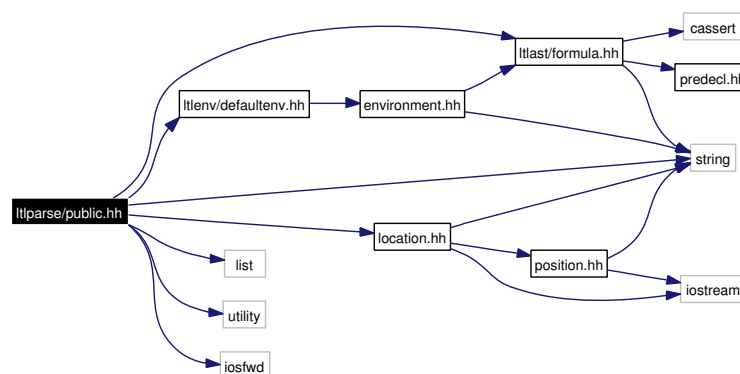
##### Returns:

`true` iff any diagnostic was output.

## 12.19 ltlparse/public.hh File Reference

```
#include "ltlast/formula.hh"
#include "location.hh"
#include "ltlenv/defaultenv.hh"
#include <string>
#include <list>
#include <utility>
#include <iosfwd>
```

Include dependency graph for public.hh:



### Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

### Typedefs

- typedef std::pair< [yy::Location](#), std::string > [parse\\_error](#)  
*A parse diagnostic with its location.*
- typedef std::list< [parse\\_error](#) > [parse\\_error\\_list](#)  
*A list of parser diagnostics, as filled by parse.*

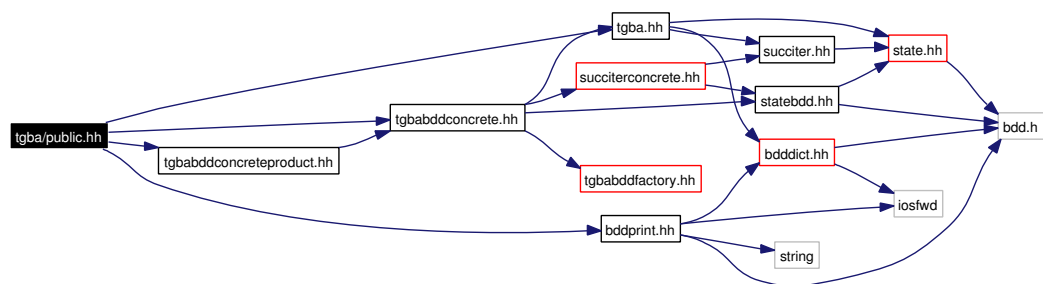
### Functions

- formula \* [parse](#) (const std::string &ltl\_string, [parse\\_error\\_list](#) &error\_list, environment &env=default\_environment::instance(), bool debug=false)  
*Build a formula from an LTL string.*
- bool [format\\_parse\\_errors](#) (std::ostream &os, const std::string &ltl\_string, [parse\\_error\\_list](#) &error\_list)  
*Format diagnostics produced by [spot::ltl::parse](#).*

## 12.20 tgba/public.hh File Reference

```
#include "tgba.hh"
#include "tgbabddconcrete.hh"
#include "tgbabddconcreteproduct.hh"
#include "bddprint.hh"
```

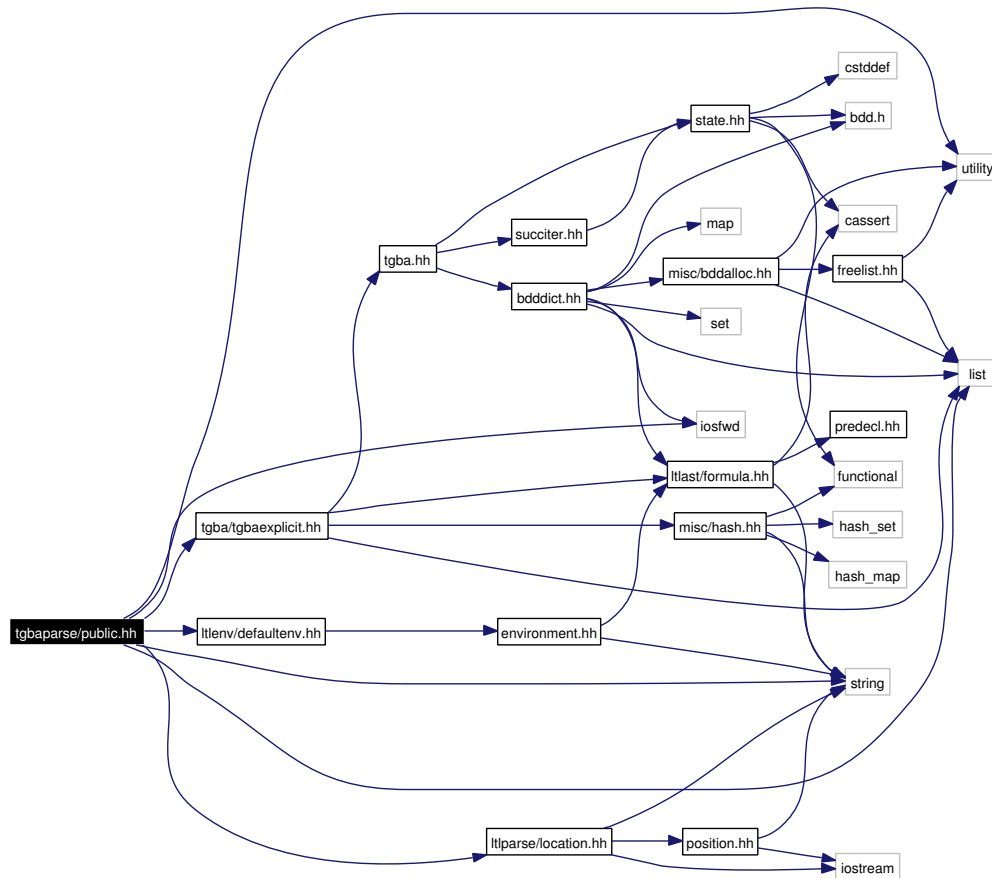
Include dependency graph for public.hh:



## 12.21 tgbaparse/public.hh File Reference

```
#include "tgba/tgbaexplicit.hh"
#include "ltlparse/location.hh"
#include "ltlenv/defaultenv.hh"
#include <string>
#include <list>
#include <utility>
#include <iosfwd>
```

Include dependency graph for public.hh:



## Namespaces

- namespace [spot](#)

## Typedefs

- typedef std::pair< [yy::Location](#), std::string > [tgba\\_parse\\_error](#)  
A parse diagnostic with its location.
- typedef std::list< [tgba\\_parse\\_error](#) > [tgba\\_parse\\_error\\_list](#)  
A list of parser diagnostics, as filled by parse.

## Functions

- [tgba\\_explicit](#) \* [tgba\\_parse](#) (const std::string &filename, [tgba\\_parse\\_error\\_list](#) &error\_list, bdd\_dict \*dict, ltl::environment &env=ltl::default\_environment::instance(), bool debug=false)  
Build a [spot::tgba\\_explicit](#) from a text file.
- bool [format\\_tgba\\_parse\\_errors](#) (std::ostream &os, const std::string &filename, [tgba\\_parse\\_error\\_list](#) &error\_list)

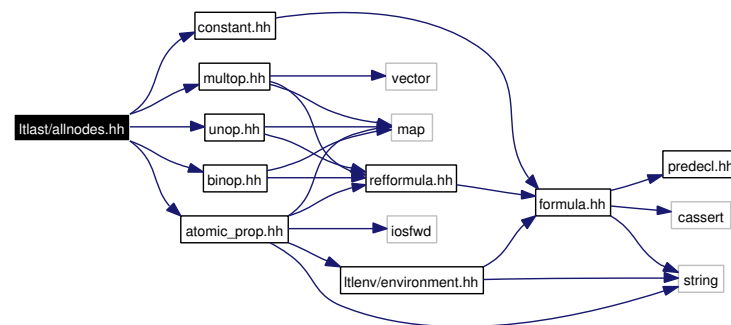
Format diagnostics produced by `spot::tgba_parse`.

## 12.22 Itlast/allnodes.hh File Reference

Define all LTL node types.

```
#include "binop.hh"
#include "unop.hh"
#include "multop.hh"
#include "atomic_prop.hh"
#include "constant.hh"
```

Include dependency graph for allnodes.hh:



### 12.22.1 Detailed Description

Define all LTL node types.

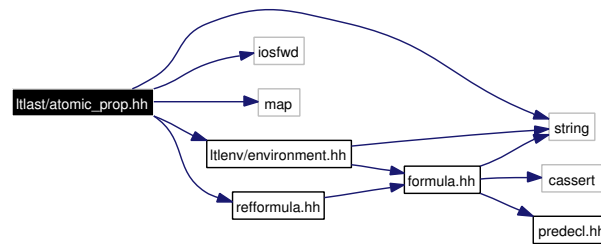
This file is usually needed when **defining** a visitor. Prefer `ltlast/predecl.hh` when only **declaring** methods and functions over LTL nodes.

## 12.23 Itlast/atomic\_prop.hh File Reference

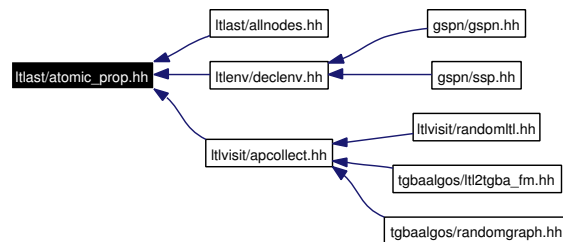
LTL atomic propositions.

```
#include <string>
#include <iosfwd>
#include <map>
#include "refformula.hh"
#include "ltlenv/environment.hh"
```

Include dependency graph for `atomic_prop.hh`:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

### 12.23.1 Detailed Description

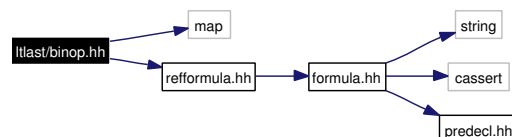
LTL atomic propositions.

## 12.24 Itlast/binop.hh File Reference

LTL binary operators.

```
#include <map>
#include "reformula.hh"
```

Include dependency graph for binop.hh:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

### 12.24.1 Detailed Description

LTL binary operators.

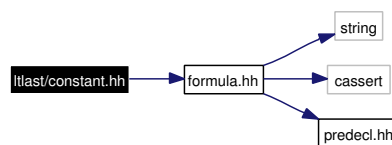
This does not include AND and OR operators. These are considered to be multi-operand operators (see [spot::ltl::multop](#)).

## 12.25 Itlast/constant.hh File Reference

LTL constants.

```
#include "formula.hh"
```

Include dependency graph for constant.hh:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

### 12.25.1 Detailed Description

LTL constants.

## 12.26 Itlast/formula.hh File Reference

LTL formula interface.

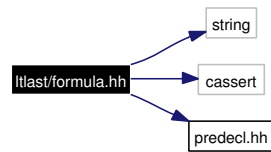
```
#include <string>
```

```
#include <cassert>
```

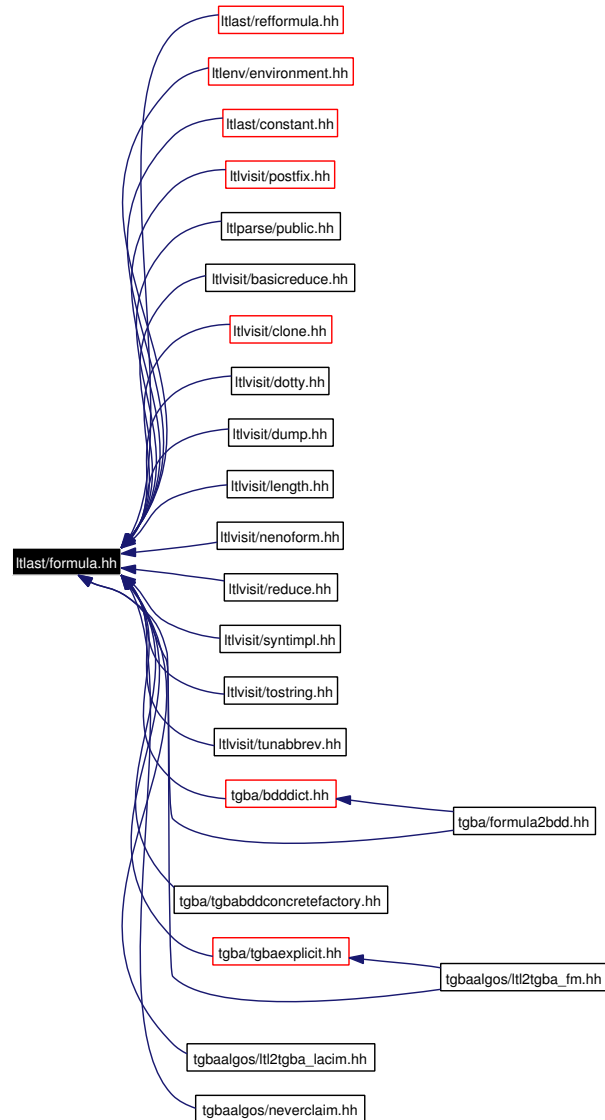
```
#include "predecl.hh"
```

Include dependency graph for formula.hh:





This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [spot](#)
- namespace [spot:Itl](#)

### 12.26.1 Detailed Description

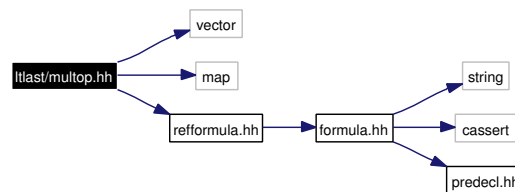
LTL formula interface.

## 12.27 Itlast/multop.hh File Reference

LTL multi-operand operators.

```
#include <vector>
#include <map>
#include "reformula.hh"
```

Include dependency graph for multop.hh:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace `spot`
- namespace `spot::ltl`

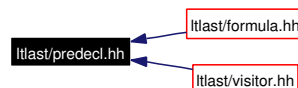
### 12.27.1 Detailed Description

LTL multi-operand operators.

## 12.28 Itlast/predecl.hh File Reference

Predeclare all LTL node types.

This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace `spot`
- namespace `spot::ltl`

### 12.28.1 Detailed Description

Predeclare all LTL node types.

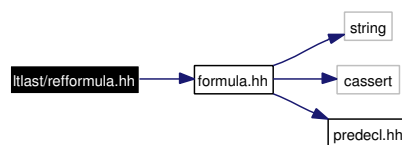
This file is usually used when **declaring** methods and functions over LTL nodes. Use `Itlast/allnodes.hh` or an individual header when the definition of the node is actually needed.

## 12.29 Itlast/reformula.hh File Reference

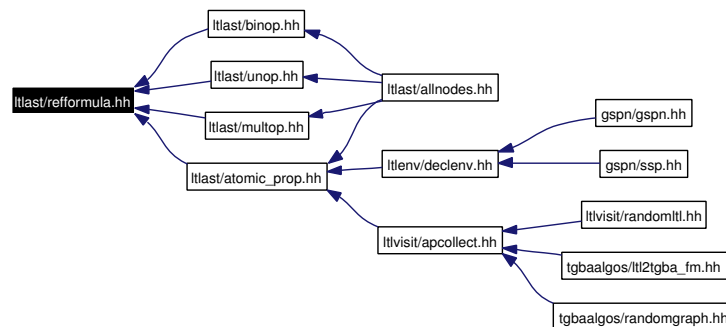
Reference-counted LTL formulae.

```
#include "formula.hh"
```

Include dependency graph for `reformula.hh`:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace `spot`
- namespace `spot::ltl`

### 12.29.1 Detailed Description

Reference-counted LTL formulae.

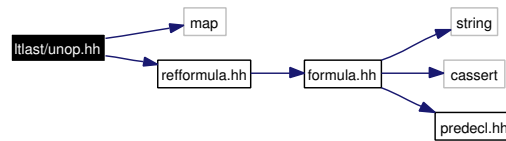
## 12.30 Itlast/unop.hh File Reference

LTL unary operators.

```
#include <map>
```

```
#include "reformula.hh"
```

Include dependency graph for `unop.hh`:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

### 12.30.1 Detailed Description

LTL unary operators.

## 12.31 Itlast/visitor.hh File Reference

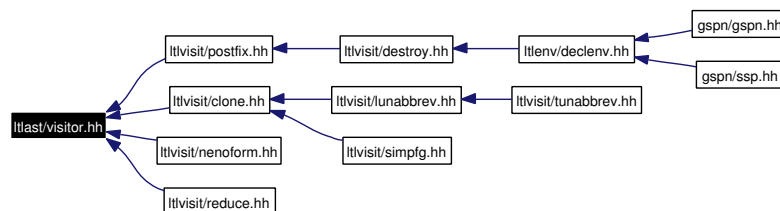
LTL visitor interface.

```
#include "predecl.hh"
```

Include dependency graph for visitor.hh:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

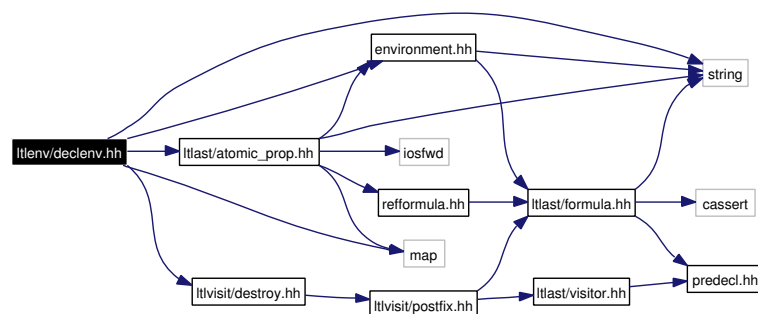
### 12.31.1 Detailed Description

LTL visitor interface.

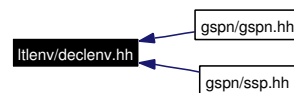
## 12.32 ltlenv/declenv.hh File Reference

```
#include "environment.hh"
#include <string>
#include <map>
#include "ltlvisit/destroy.hh"
#include "ltlast/atomic_prop.hh"
```

Include dependency graph for declenv.hh:



This graph shows which files directly or indirectly include this file:



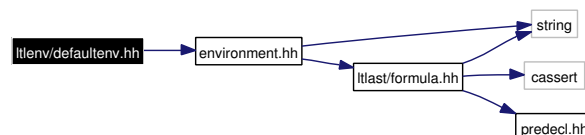
### Namespaces

- namespace [spot](#)
- namespace [spot:ltl](#)

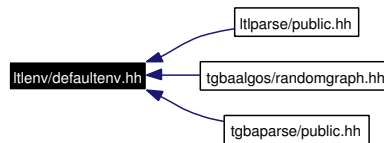
## 12.33 ltlenv/defaultenv.hh File Reference

```
#include "environment.hh"
```

Include dependency graph for defaultenv.hh:



This graph shows which files directly or indirectly include this file:



### Namespaces

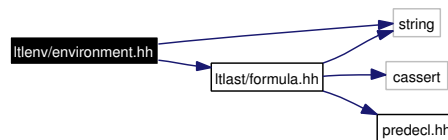
- namespace [spot](#)
- namespace [spot::ltl](#)

## 12.34 Itlenv/environment.hh File Reference

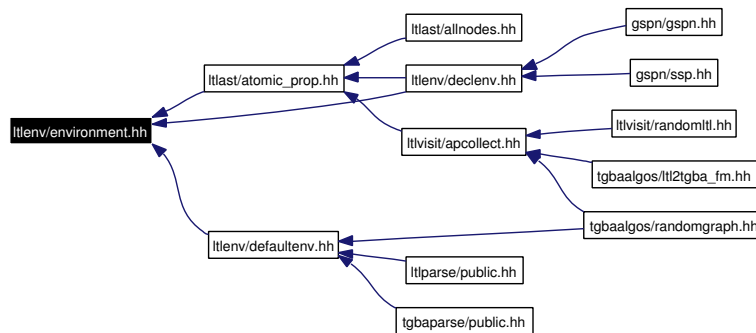
```
#include "ltlast/formula.hh"
```

```
#include <string>
```

Include dependency graph for environment.hh:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

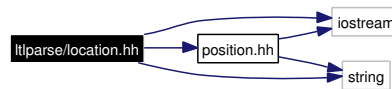
## 12.35 Itlparse/location.hh File Reference

```
#include <iostream>
```

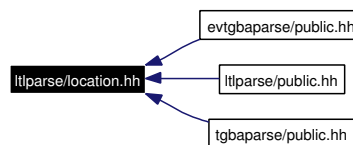
```
#include <string>
```

```
#include "position.hh"
```

Include dependency graph for location.hh:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [yy](#)

## Functions

- `const Location operator+ (const Location &begin, const Location &end)`  
*Join two [Location](#) objects to create a [Location](#).*
- `const Location operator+ (const Location &begin, unsigned int width)`  
*Add two [Location](#) objects.*
- `Location & operator+= (Location &res, unsigned int width)`  
*Add and assign a [Location](#).*
- `std::ostream & operator<< (std::ostream &ostr, const Location &loc)`  
*Intercept output stream redirection.*

### 12.35.1 Detailed Description

Define the Location class.

### 12.35.2 Function Documentation

#### 12.35.2.1 `const Location operator+ (const Location & begin, unsigned int width)` [`inline`]

Add two [Location](#) objects.

#### 12.35.2.2 `const Location operator+ (const Location & begin, const Location & end)` [`inline`]

Join two [Location](#) objects to create a [Location](#).

**12.35.2.3 Location& operator+= (Location &res, unsigned int width) [inline]**

Add and assign a [Location](#).

**12.35.2.4 std::ostream& operator<< (std::ostream &ostr, const Location &loc) [inline]**

Intercept output stream redirection.

**Parameters:**

*ostr* the destination output stream

*loc* a reference to the [Location](#) to redirect

Avoid duplicate information.

**12.36 Itlparse/position.hh File Reference**

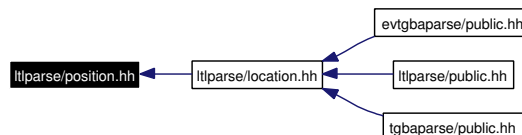
```
#include <iostream>
```

```
#include <string>
```

Include dependency graph for position.hh:



This graph shows which files directly or indirectly include this file:

**Namespaces**

- namespace [yy](#)

**Functions**

- const Position & [operator+=](#) (Position &res, const int width)  
Add and assign a [Position](#).
- const Position [operator+](#) (const Position &begin, const int width)  
Add two [Position](#) objects.
- const Position & [operator-=](#) (Position &res, const int width)  
Add and assign a [Position](#).
- const Position [operator-](#) (const Position &begin, const int width)



Add two [Position](#) objects.

- `std::ostream & operator<< (std::ostream &ostr, const Position &pos)`

Intercept output stream redirection.

### 12.36.1 Detailed Description

Define the Location class.

### 12.36.2 Function Documentation

#### 12.36.2.1 `const Position operator+ (const Position & begin, const int width)` [inline]

Add two [Position](#) objects.

#### 12.36.2.2 `const Position& operator+= (Position & res, const int width)` [inline]

Add and assign a [Position](#).

#### 12.36.2.3 `const Position operator- (const Position & begin, const int width)` [inline]

Add two [Position](#) objects.

#### 12.36.2.4 `const Position& operator-= (Position & res, const int width)` [inline]

Add and assign a [Position](#).

#### 12.36.2.5 `std::ostream& operator<< (std::ostream & ostr, const Position & pos)` [inline]

Intercept output stream redirection.

#### Parameters:

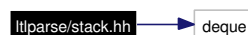
*ostr* the destination output stream

*pos* a reference to the [Position](#) to redirect

## 12.37 Itlparse/stack.hh File Reference

```
#include <deque>
```

Include dependency graph for stack.hh:



### Namespaces

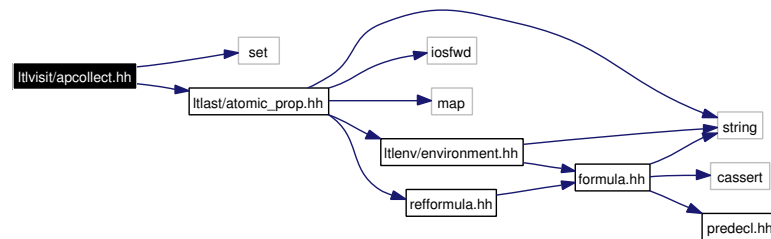
- namespace [yy](#)

## 12.38 Itlvisit/apcollect.hh File Reference

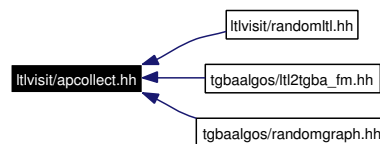
```
#include <set>
```

```
#include "ltlast/atomic_prop.hh"
```

Include dependency graph for apcollect.hh:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

### Typedefs

- typedef `std::set< atomic_prop *, formula_ptr_less_than >` [atomic\\_prop\\_set](#)  
*Set of atomic propositions.*

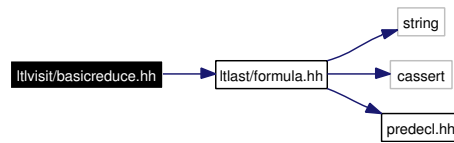
### Functions

- [atomic\\_prop\\_set](#) \* [atomic\\_prop\\_collect](#) (const formula \*f, [atomic\\_prop\\_set](#) \*s=0)  
*Return the set of atomic propositions occurring in a formula.*

## 12.39 Itlvisit/basicreduce.hh File Reference

```
#include "ltlast/formula.hh"
```

Include dependency graph for basicreduce.hh:



## Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

## Functions

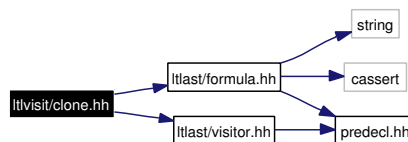
- formula \* [basic\\_reduce](#) (const formula \*f)  
*Basic rewritings.*
- bool [is\\_GF](#) (const formula \*f)  
*Whether a formula starts with GF.*
- bool [is\\_FG](#) (const formula \*f)  
*Whether a formula starts with FG.*

## 12.40 Itlvisit/clone.hh File Reference

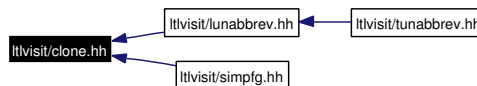
```
#include "ltlast/formula.hh"
```

```
#include "ltlast/visitor.hh"
```

Include dependency graph for clone.hh:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

**Functions**

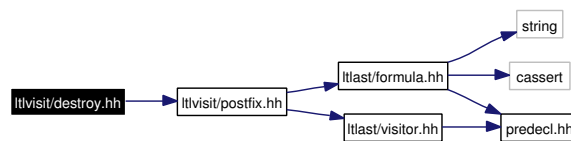
- formula \* [clone](#) (const formula \*f)

*Clone a formula.*

**12.41 ltlvisit/destroy.hh File Reference**

```
#include "ltlvisit/postfix.hh"
```

Include dependency graph for destroy.hh:



This graph shows which files directly or indirectly include this file:

**Namespaces**

- namespace [spot](#)
- namespace [spot::ltl](#)

**Functions**

- void [destroy](#) (const formula \*f)

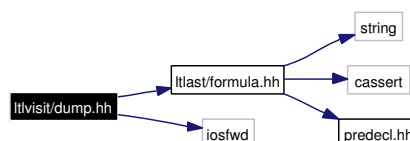
*Destroys a formula.*

**12.42 ltlvisit/dump.hh File Reference**

```
#include "ltlast/formula.hh"
```

```
#include <iosfwd>
```

Include dependency graph for dump.hh:



### Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

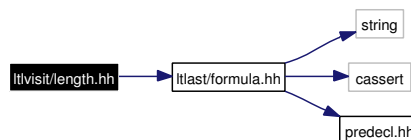
### Functions

- `std::ostream & dump (std::ostream &os, const formula *f)`  
*Dump a formula tree.*

## 12.43 Itlvisit/length.hh File Reference

```
#include "ltlast/formula.hh"
```

Include dependency graph for length.hh:



### Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

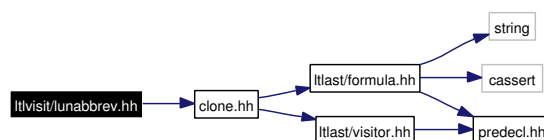
### Functions

- `int length (const formula *f)`  
*Compute the length of a formula.*

## 12.44 Itlvisit/lunabbrev.hh File Reference

```
#include "clone.hh"
```

Include dependency graph for lunabbrev.hh:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

## Functions

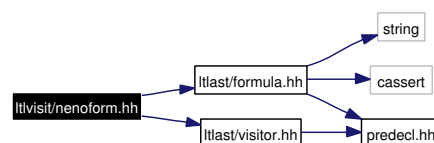
- formula \* [unabbreviate\\_logic](#) (const formula \*f)  
*Clone and rewrite a formula to remove most of the abbreviated logical operators.*

## 12.45 Itlvisit/venoform.hh File Reference

```
#include "ltlast/formula.hh"
```

```
#include "ltlast/visitor.hh"
```

Include dependency graph for venoform.hh:



## Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

## Functions

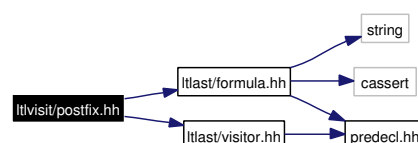
- formula \* [negative\\_normal\\_form](#) (const formula \*f, bool negated=false)  
*Build the negative normal form of f.*

## 12.46 Itlvisit/postfix.hh File Reference

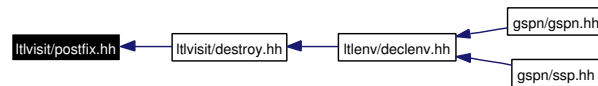
```
#include "ltlast/formula.hh"
```

```
#include "ltlast/visitor.hh"
```

Include dependency graph for postfix.hh:



This graph shows which files directly or indirectly include this file:



### Namespaces

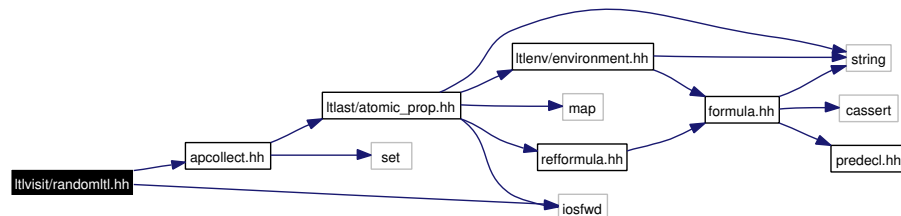
- namespace `spot`
- namespace `spot::ltl`

## 12.47 Itlvisit/randomltl.hh File Reference

```
#include "apcollect.hh"
```

```
#include <iosfwd>
```

Include dependency graph for randomltl.hh:



### Namespaces

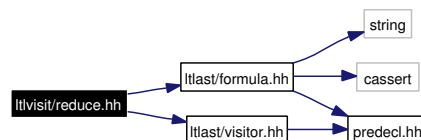
- namespace `spot`
- namespace `spot::ltl`

## 12.48 Itlvisit/reduce.hh File Reference

```
#include "ltlast/formula.hh"
```

```
#include "ltlast/visitor.hh"
```

Include dependency graph for reduce.hh:



### Namespaces

- namespace `spot`
- namespace `spot::ltl`

## Enumerations

- enum `reduce_options` {  
`Reduce_None` = 0, `Reduce_Basics` = 1, `Reduce_Syntactic_Implications` = 2, `Reduce_Eventuality_And_Universality` = 4,  
`Reduce_All` = -1U }  
*Options for `spot::ltl::reduce`.*

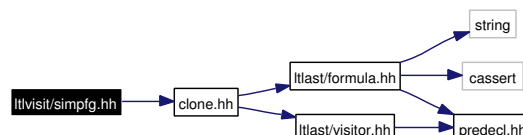
## Functions

- formula \* `reduce` (const formula \*f, int opt=`Reduce_All`)  
*Reduce a formula f.*
- bool `is_eventual` (const formula \*f)  
*Check whether a formula is a pure eventuality.*
- bool `is_universal` (const formula \*f)  
*Check whether a formula is purely universal.*

## 12.49 Itlvisit/simpfg.hh File Reference

```
#include "clone.hh"
```

Include dependency graph for simpfg.hh:



## Namespaces

- namespace `spot`
- namespace `spot::ltl`

## Functions

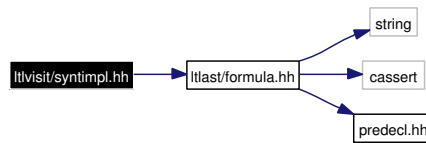
- formula \* `simplify_f_g` (const formula \*f)  
*Replace true U f and false R g by F f and G g.*

## 12.50 Itlvisit/syntimpl.hh File Reference

```
#include "ltlast/formula.hh"
```

Include dependency graph for syntimpl.hh:





## Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

## Functions

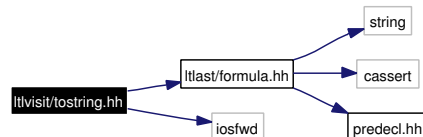
- bool [syntactic\\_implication](#) (const formula \*f1, const formula \*f2)  
*Syntactic implication.*
- bool [syntactic\\_implication\\_neg](#) (const formula \*f1, const formula \*f2, bool right)  
*Syntactic implication.*

## 12.51 ltlvisit/tostring.hh File Reference

```
#include <ltlast/formula.hh>
```

```
#include <iosfwd>
```

Include dependency graph for tostring.hh:



## Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

## Functions

- std::ostream & [to\\_string](#) (const formula \*f, std::ostream &os)  
*Output a formula as a (parsable) string.*
- std::string [to\\_string](#) (const formula \*f)  
*Convert a formula into a (parsable) string.*
- std::ostream & [to\\_spin\\_string](#) (const formula \*f, std::ostream &os)  
*Output a formula as a (parsable by Spin) string.*

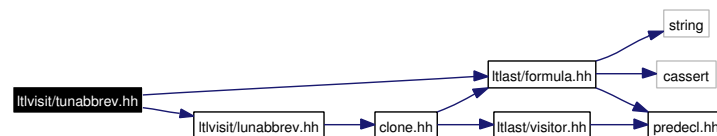
- `std::string to_spin_string (const formula *f)`  
Convert a formula into a (parsable by Spin) string.

## 12.52 ltlvisit/tunabbrev.hh File Reference

```
#include "ltlast/formula.hh"
```

```
#include "ltlvisit/lunabbrev.hh"
```

Include dependency graph for tunabbrev.hh:



### Namespaces

- namespace `spot`
- namespace `spot::ltl`

### Functions

- `formula * unabbreviate_ltl (const formula *f)`  
Clone and rewrite a formula to remove most of the abbreviated LTL and logical operators.

#### 12.52.1 Function Documentation

##### 12.52.1.1 `formula* unabbreviate_ltl (const formula *f)`

Clone and rewrite a formula to remove most of the abbreviated LTL and logical operators.

The rewriting performed on logical operator is the same as the one done by `spot::ltl::unabbreviate_logic`.

This will also rewrite unary operators such as `unop::F`, and `unop::G`, using only `binop::U`, and `binop::R`.

## 12.53 misc/bareword.hh File Reference

```
#include <string>
```

Include dependency graph for bareword.hh:



### Namespaces

- namespace `spot`

**Functions**

- bool `is_bare_word` (const char \*str)
- std::string `quote_unless_bare_word` (const std::string &str)

*Double-quote words that are not bare.*

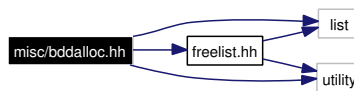
**12.54 misc/bddalloc.hh File Reference**

```
#include "freelist.hh"
```

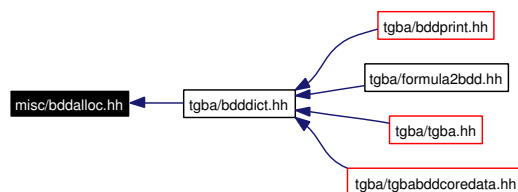
```
#include <list>
```

```
#include <utility>
```

Include dependency graph for bddalloc.hh:



This graph shows which files directly or indirectly include this file:

**Namespaces**

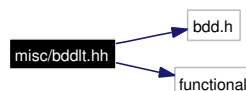
- namespace `spot`

**12.55 misc/bddlt.hh File Reference**

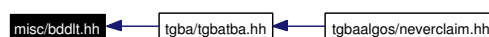
```
#include <bdd.h>
```

```
#include <functional>
```

Include dependency graph for bddlt.hh:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [spot](#)

## 12.56 misc/escape.hh File Reference

```
#include <iosfwd>
```

```
#include <string>
```

Include dependency graph for escape.hh:



## Namespaces

- namespace [spot](#)

## Functions

- `std::ostream & escape\_str (std::ostream &os, const std::string &str)`  
*Escape " and \ characters in str.*
- `std::string escape\_str (const std::string &str)`  
*Escape " and \ characters in str.*

## 12.57 misc/freelist.hh File Reference

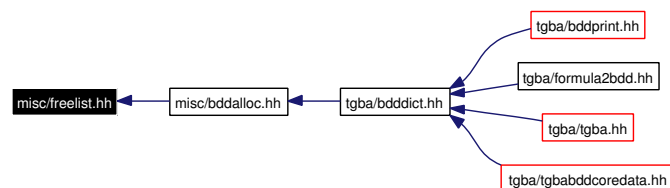
```
#include <list>
```

```
#include <utility>
```

Include dependency graph for freelist.hh:



This graph shows which files directly or indirectly include this file:



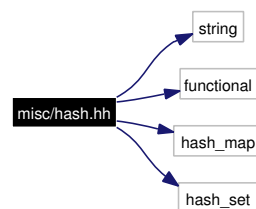
## Namespaces

- namespace **spot**

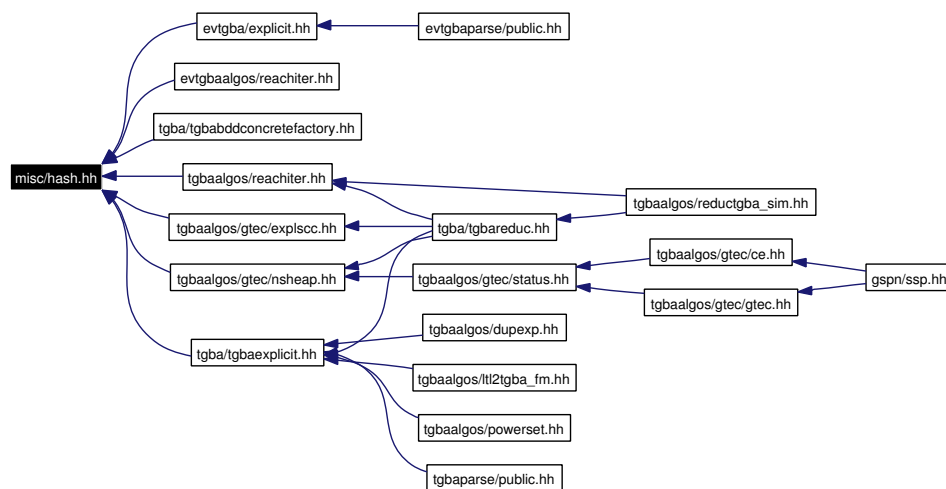
## 12.58 misc/hash.hh File Reference

```
#include <string>
#include <functional>
#include <hash_map>
#include <hash_set>
```

Include dependency graph for hash.hh:



This graph shows which files directly or indirectly include this file:



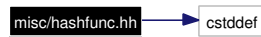
## Namespaces

- namespace **spot**

## 12.59 misc/hashfunc.hh File Reference

```
#include <cstdint>
```

Include dependency graph for hashfunc.hh:



## Namespaces

- namespace [spot](#)

## Functions

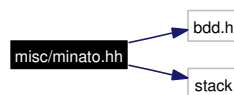
- `size_t wang32_hash` (`size_t` key)  
*Thomas Wang's 32 bit hash function.*

## 12.60 misc/minato.hh File Reference

```
#include <bdd.h>
```

```
#include <stack>
```

Include dependency graph for minato.hh:



## Namespaces

- namespace [spot](#)

## 12.61 misc/modgray.hh File Reference

### Namespaces

- namespace [spot](#)

## 12.62 misc/random.hh File Reference

```
#include <cmath>
```

Include dependency graph for random.hh:



## Namespaces

- namespace [spot](#)

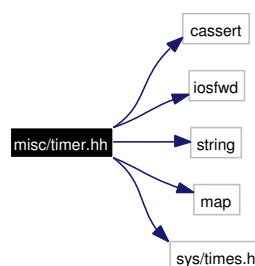
## Functions

- void [srand](#) (unsigned int seed)  
*Reset the seed of the pseudo-random number generator.*
- int [rrand](#) (int min, int max)  
*Compute a pseudo-random integer value between min and max included.*
- int [mrand](#) (int max)  
*Compute a pseudo-random integer value between 0 and max-1 included.*
- double [drand](#) ()  
*Compute a pseudo-random double value between 0.0 and 1.0 (1.0 excluded).*
- double [nrand](#) ()  
*Compute a pseudo-random double value following a standard normal distribution. (Odeh & Evans).*
- double [bmrnd](#) ()  
*Compute a pseudo-random double value following a standard normal distribution. (Box-Muller).*
- int [prand](#) (double p)  
*Return a pseudo-random positive integer value following a Poisson distribution with parameter p.*

## 12.63 misc/timer.hh File Reference

```
#include <cassert>
#include <iosfwd>
#include <string>
#include <map>
#include <sys/times.h>
```

Include dependency graph for timer.hh:



## Namespaces

- namespace [spot](#)

## 12.64 misc/version.hh File Reference

### Namespaces

- namespace [spot](#)

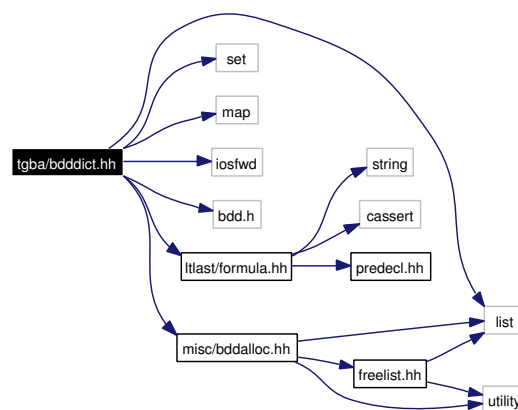
### Functions

- const char \* [version](#) ()  
*Return Spot's version.*

## 12.65 tgba/bdddict.hh File Reference

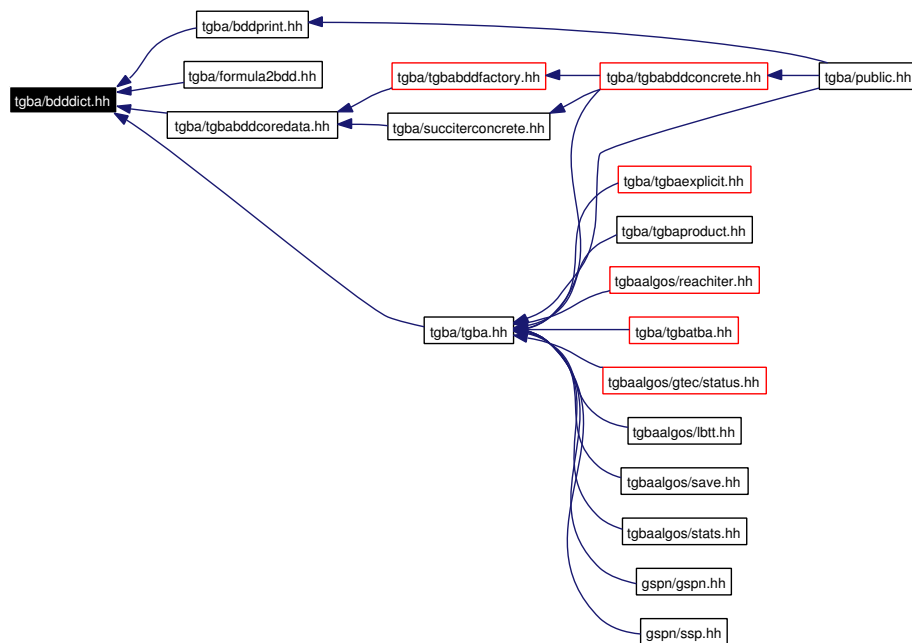
```
#include <list>
#include <set>
#include <map>
#include <iosfwd>
#include <bdd.h>
#include "ltlast/formula.hh"
#include "misc/bddalloc.hh"
```

Include dependency graph for bdddict.hh:



This graph shows which files directly or indirectly include this file:





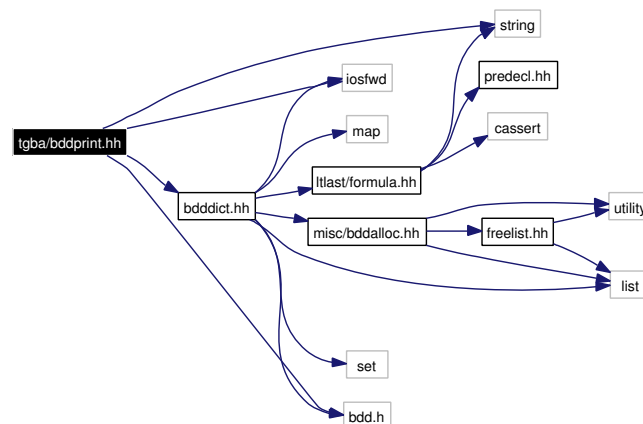
## Namespaces

- namespace [spot](#)

## 12.66 tgba/bddprint.hh File Reference

```
#include <string>
#include <iosfwd>
#include "bdddict.hh"
#include <bdd.h>
```

Include dependency graph for bddprint.hh:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [spot](#)

## Functions

- `std::ostream & bdd\_print\_sat (std::ostream &os, const bdd_dict *dict, bdd b)`  
*Print a BDD as a list of literals.*
- `std::string bdd\_format\_sat (const bdd_dict *dict, bdd b)`  
*Format a BDD as a list of literals.*
- `std::ostream & bdd\_print\_acc (std::ostream &os, const bdd_dict *dict, bdd b)`  
*Print a BDD as a list of acceptance conditions.*
- `std::ostream & bdd\_print\_accset (std::ostream &os, const bdd_dict *dict, bdd b)`  
*Print a BDD as a set of acceptance conditions.*
- `std::string bdd\_format\_accset (const bdd_dict *dict, bdd b)`  
*Format a BDD as a set of acceptance conditions.*
- `std::ostream & bdd\_print\_set (std::ostream &os, const bdd_dict *dict, bdd b)`  
*Print a BDD as a set.*
- `std::string bdd\_format\_set (const bdd_dict *dict, bdd b)`  
*Format a BDD as a set.*
- `std::ostream & bdd\_print\_formula (std::ostream &os, const bdd_dict *dict, bdd b)`  
*Print a BDD as a formula.*
- `std::string bdd\_format\_formula (const bdd_dict *dict, bdd b)`  
*Format a BDD as a formula.*
- `std::ostream & bdd\_print\_dot (std::ostream &os, const bdd_dict *dict, bdd b)`  
*Print a BDD as a diagram in doty format.*
- `std::ostream & bdd\_print\_table (std::ostream &os, const bdd_dict *dict, bdd b)`  
*Print a BDD as a table.*

### 12.66.1 Function Documentation

#### 12.66.1.1 `std::string bdd_format_accset (const bdd_dict *dict, bdd b)`

Format a BDD as a set of acceptance conditions.

This is used when saving a TGBA.

**Parameters:**

*dict* The dictionary to use, to lookup variables.

*b* The BDD to print.

**Returns:**

The BDD formatted as a string.

**12.66.1.2 std::string bdd\_format\_formula (const bdd\_dict \* dict, bdd b)**

Format a BDD as a formula.

**Parameters:**

*dict* The dictionary to use, to lookup variables.

*b* The BDD to print.

**Returns:**

The BDD formatted as a string.

**12.66.1.3 std::string bdd\_format\_sat (const bdd\_dict \* dict, bdd b)**

Format a BDD as a list of literals.

This assumes that *b* is a conjunction of literals.

**Parameters:**

*dict* The dictionary to use, to lookup variables.

*b* The BDD to print.

**Returns:**

The BDD formatted as a string.

**12.66.1.4 std::string bdd\_format\_set (const bdd\_dict \* dict, bdd b)**

Format a BDD as a set.

**Parameters:**

*dict* The dictionary to use, to lookup variables.

*b* The BDD to print.

**Returns:**

The BDD formatted as a string.

**12.66.1.5 std::ostream& bdd\_print\_acc (std::ostream & os, const bdd\_dict \* dict, bdd b)**

Print a BDD as a list of acceptance conditions.

This is used when saving a TGBA.

**Parameters:**

*os* The output stream.

*dict* The dictionary to use, to lookup variables.

*b* The BDD to print.

**Returns:**

The BDD formatted as a string.

**12.66.1.6 std::ostream& bdd\_print\_accset (std::ostream & *os*, const bdd\_dict \* *dict*, bdd *b*)**

Print a BDD as a set of acceptance conditions.

This is used when saving a TGBA.

**Parameters:**

*os* The output stream.

*dict* The dictionary to use, to lookup variables.

*b* The BDD to print.

**Returns:**

The BDD formatted as a string.

**12.66.1.7 std::ostream& bdd\_print\_dot (std::ostream & *os*, const bdd\_dict \* *dict*, bdd *b*)**

Print a BDD as a diagram in doty format.

**Parameters:**

*os* The output stream.

*dict* The dictionary to use, to lookup variables.

*b* The BDD to print.

**12.66.1.8 std::ostream& bdd\_print\_formula (std::ostream & *os*, const bdd\_dict \* *dict*, bdd *b*)**

Print a BDD as a formula.

**Parameters:**

*os* The output stream.

*dict* The dictionary to use, to lookup variables.

*b* The BDD to print.

**12.66.1.9 std::ostream& bdd\_print\_sat (std::ostream & *os*, const bdd\_dict \* *dict*, bdd *b*)**

Print a BDD as a list of literals.

This assumes that *b* is a conjunction of literals.

**Parameters:**

*os* The output stream.

*dict* The dictionary to use, to lookup variables.

*b* The BDD to print.

**12.66.1.10** `std::ostream& bdd_print_set (std::ostream & os, const bdd_dict * dict, bdd b)`

Print a BDD as a set.

### Parameters:

*os* The output stream.

*dict* The dictionary to use, to lookup variables.

***b*** The BDD to print.

### 12.66.1.11 `std::ostream& bdd_print_table (std::ostream & os, const bdd_dict * dict, bdd b)`

Print a BDD as a table.

### Parameters:

*os* The output stream.

*dict* The dictionary to use, to lookup variables.

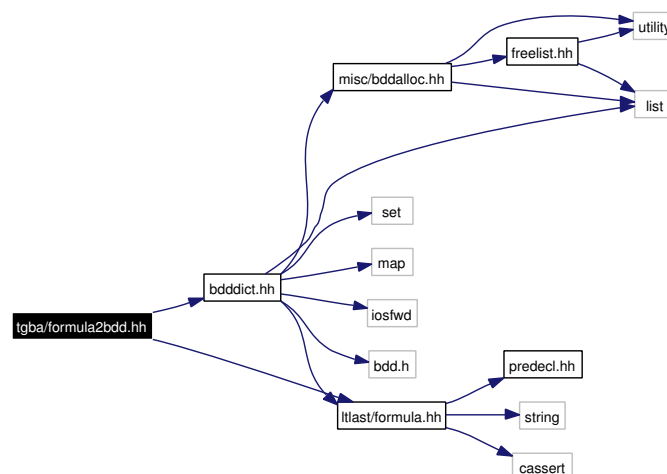
***b*** The BDD to print.

## 12.67 tgba/formula2bdd.hh File Reference

```
#include "bdddict.hh"
```

```
#include "ltlast/formula.hh"
```

Include dependency graph for formula2bdd.hh:



## Namespaces

- namespace **spot**

## Functions

- `bdd formula_to_bdd` (const ltl::formula \*f, bdd\_dict \*d, void \*for\_me)
- `const ltl::formula * bdd_to_formula` (bdd f, const bdd\_dict \*d)

### 12.67.1 Function Documentation

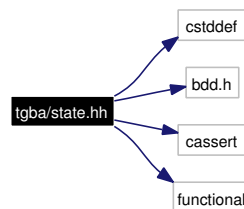
12.67.1.1 `const ltl::formula* bdd_to_formula (bdd f, const bdd_dict * d)`

12.67.1.2 `bdd formula_to_bdd (const ltl::formula * f, bdd_dict * d, void * for_me)`

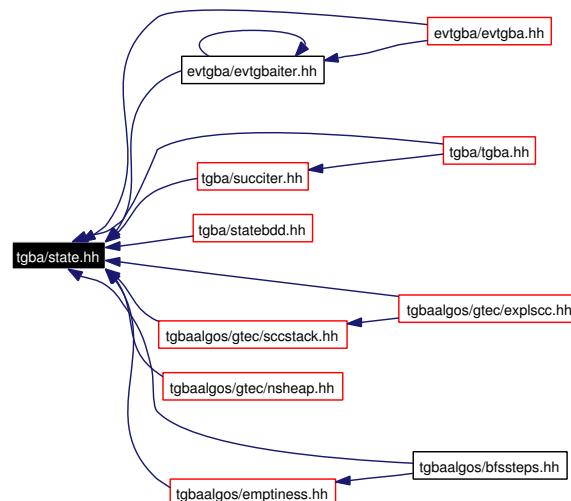
## 12.68 tgba/state.hh File Reference

```
#include <cstdlib>
#include <bdd.h>
#include <cassert>
#include <functional>
```

Include dependency graph for state.hh:



This graph shows which files directly or indirectly include this file:



## Namespaces

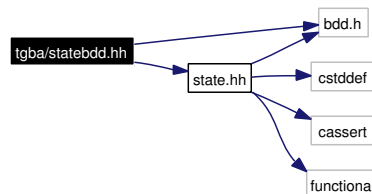
- namespace [spot](#)

## 12.69 tgba/statebdd.hh File Reference

```
#include <bdd.h>
```

```
#include "state.hh"
```

Include dependency graph for statebdd.hh:



This graph shows which files directly or indirectly include this file:



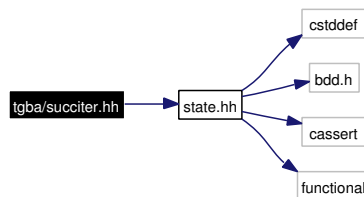
### Namespaces

- namespace `spot`

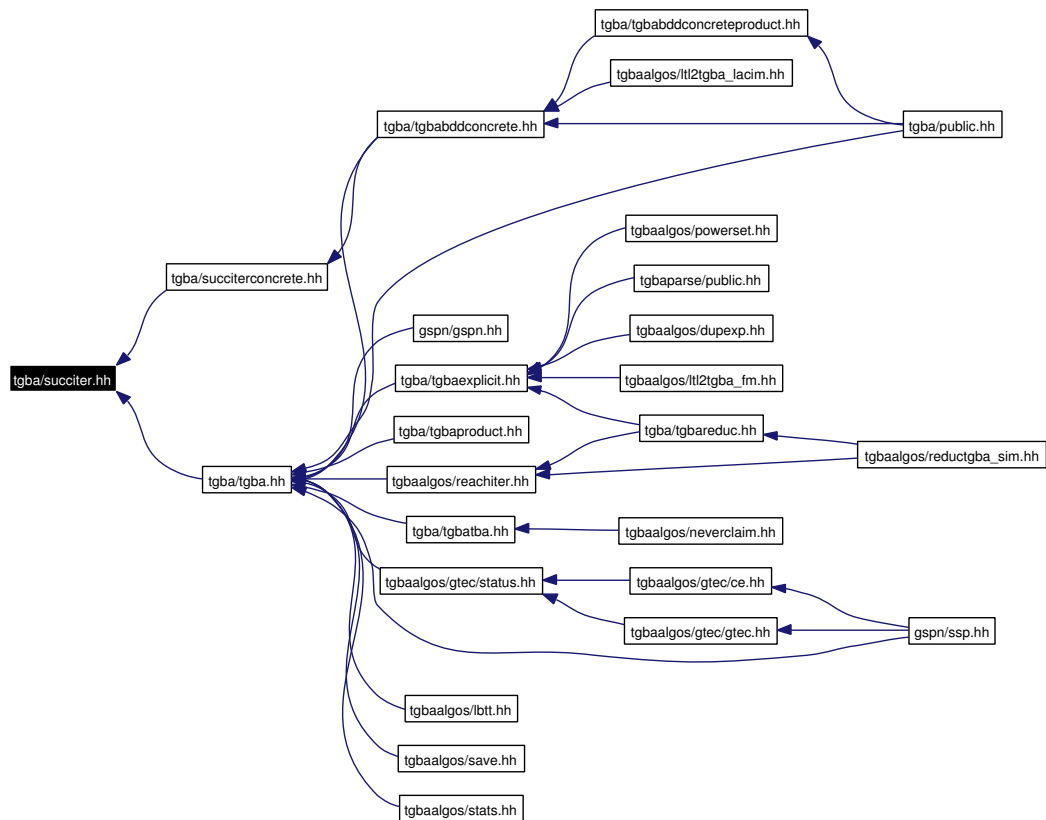
## 12.70 tgba/succiter.hh File Reference

```
#include "state.hh"
```

Include dependency graph for succiter.hh:



This graph shows which files directly or indirectly include this file:



## Namespaces

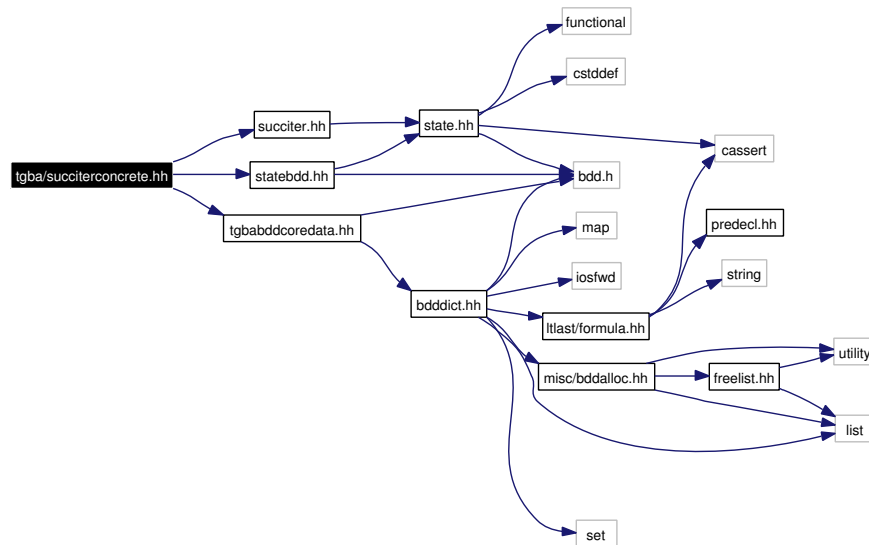
- namespace [spot](#)

## 12.71 tgba/succiterconcrete.hh File Reference

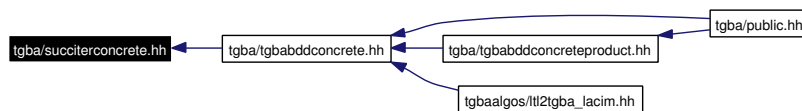
```
#include "statebdd.hh"
#include "succiter.hh"
#include "tgbabddcoredata.hh"
```

Include dependency graph for succiterconcrete.hh:





This graph shows which files directly or indirectly include this file:



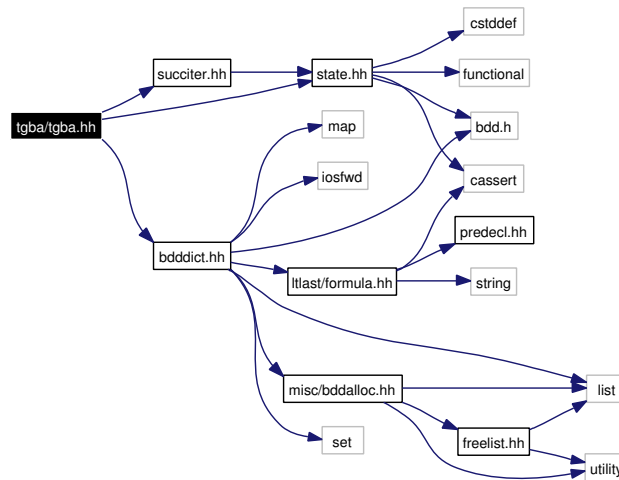
## Namespaces

- namespace [spot](#)

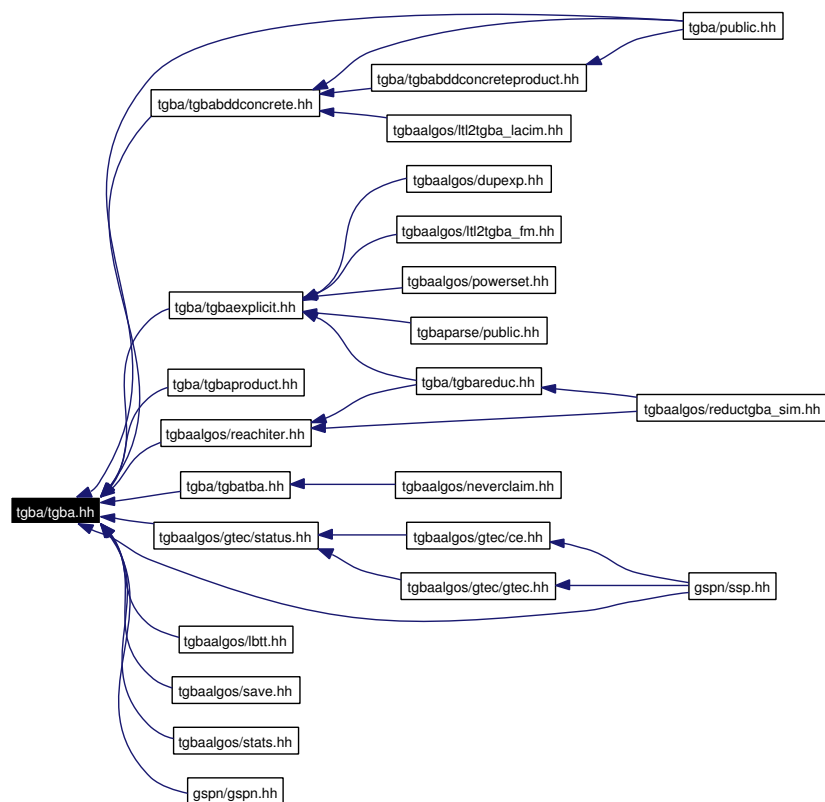
## 12.72 tgba/tgba.hh File Reference

```
#include "state.hh"
#include "succiter.hh"
#include "bdddict.hh"
```

Include dependency graph for tgba.hh:



This graph shows which files directly or indirectly include this file:



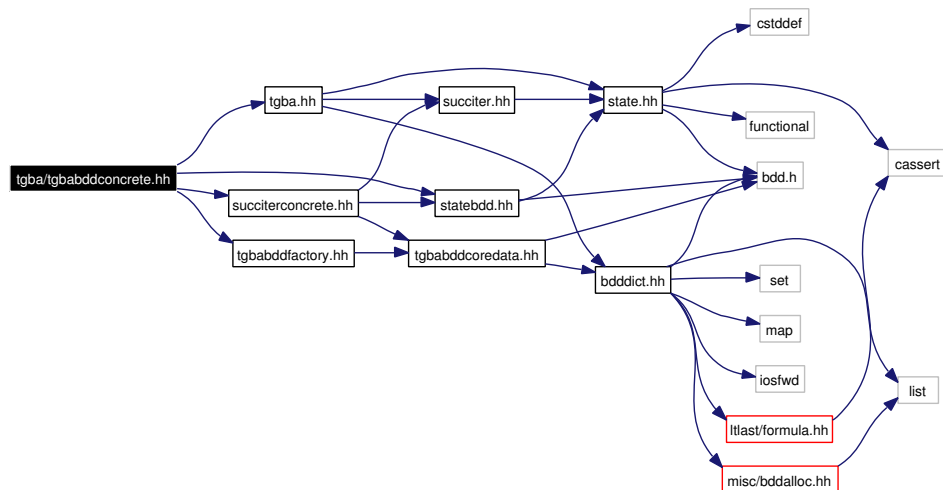
## Namespaces

- namespace [spot](#)

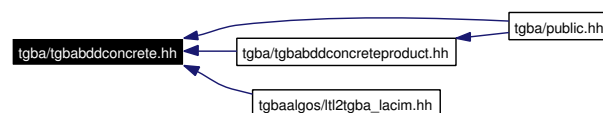
### 12.73 tgba/tgbabddconcrete.hh File Reference

```
#include "tgba.hh"
#include "statebdd.hh"
#include "tgbabddfactory.hh"
#include "succiterconcrete.hh"
```

Include dependency graph for `tgbabddconcrete.hh`:



This graph shows which files directly or indirectly include this file:



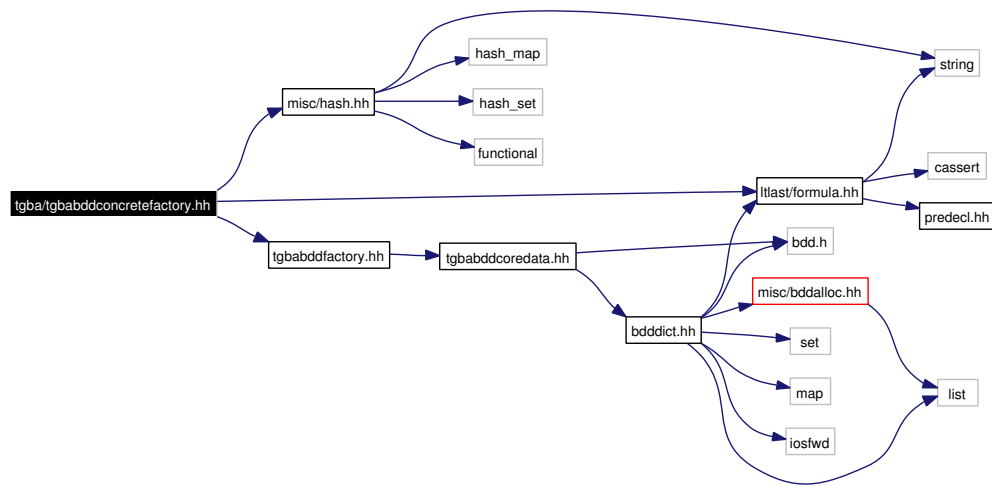
## Namespaces

- namespace **spot**

## 12.74 tgba/tgbabddconcretefactory.hh File Reference

```
#include "misc/hash.hh"
#include "ltlast/formula.hh"
#include "tgbabddfactory.hh"
```

Include dependency graph for `tgbabddconcretefactory.hh`:



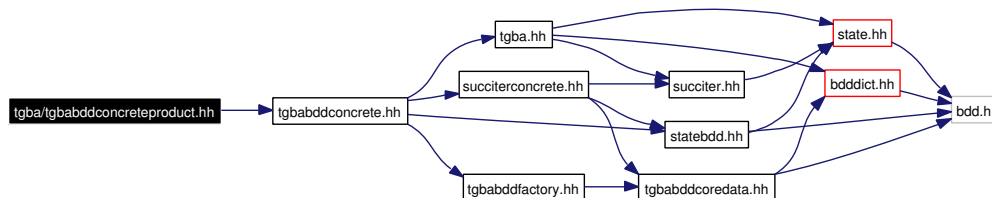
## Namespaces

- namespace [spot](#)

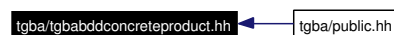
## 12.75 tgba/tgbabddconcreteproduct.hh File Reference

```
#include "tgbabddconcrete.hh"
```

Include dependency graph for tgba/tgbabddconcreteproduct.hh:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [spot](#)

## Functions

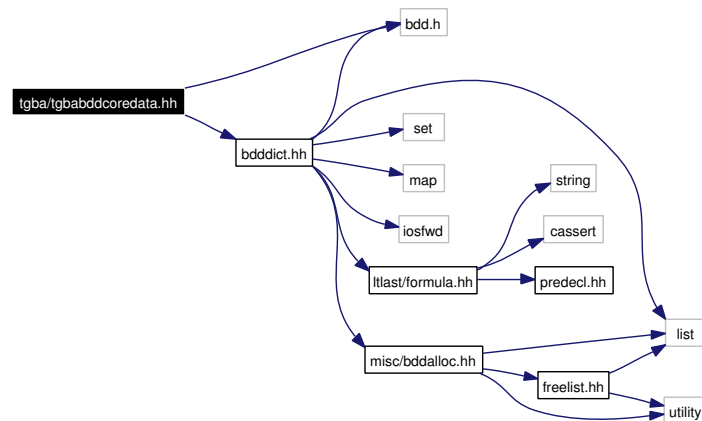
- tgba\_bdd\_concrete \* [product](#) (const tgba\_bdd\_concrete \*left, const tgba\_bdd\_concrete \*right)  
Multiplies two [spot::tgba\\_bdd\\_concrete](#) automata.

## 12.76 tgba/tgbabddcoredata.hh File Reference

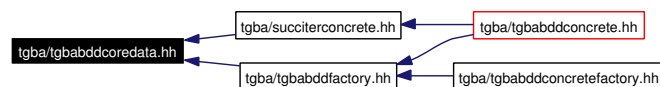
```
#include <bdd.h>
```

```
#include "bdddict.hh"
```

Include dependency graph for tgbabddcoredata.hh:



This graph shows which files directly or indirectly include this file:



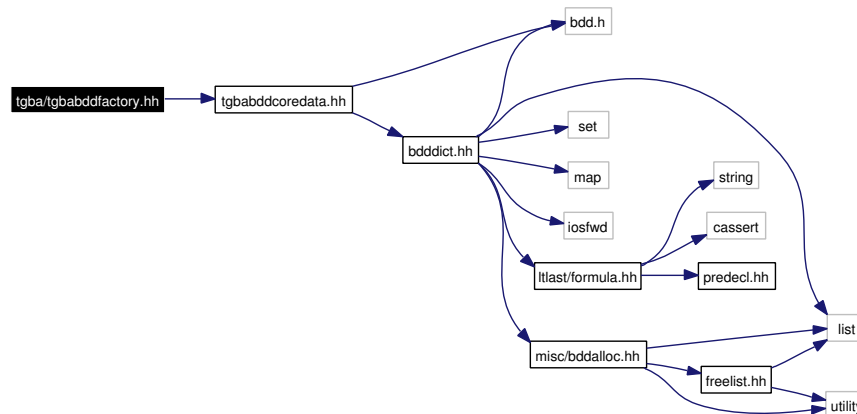
### Namespaces

- namespace `spot`

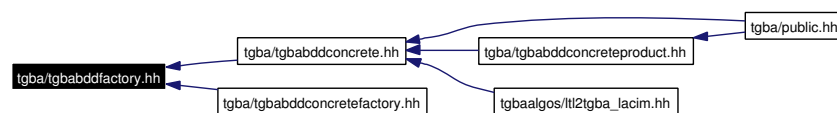
## 12.77 tgba/tgbabddfacy.hh File Reference

```
#include "tgbabddcoredata.hh"
```

Include dependency graph for tgbabddfacy.hh:



This graph shows which files directly or indirectly include this file:



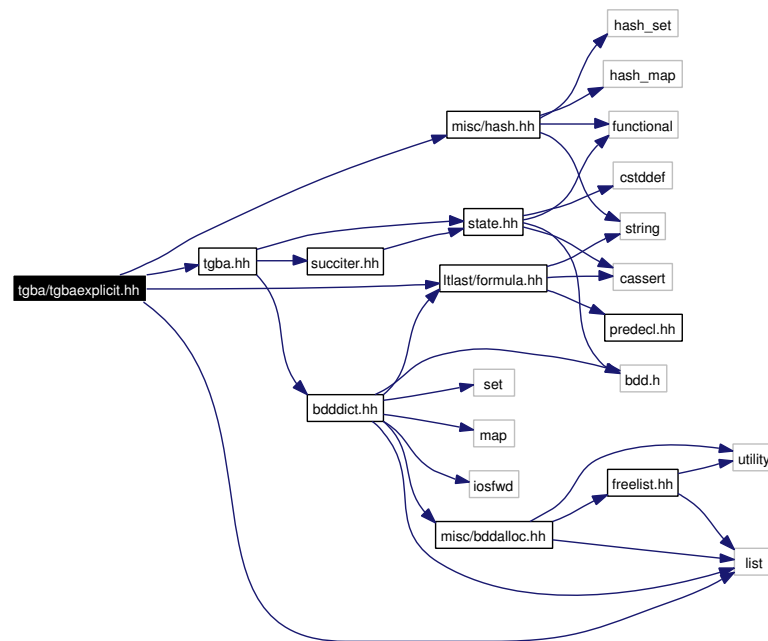
## Namespaces

- namespace [spot](#)

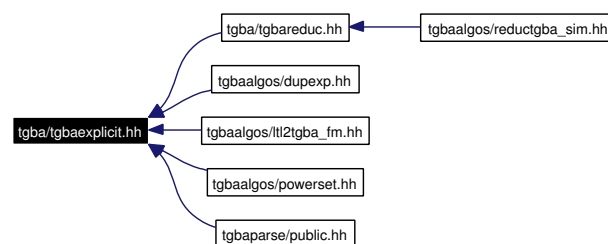
## 12.78 tgba/tgbaexplicit.hh File Reference

```
#include "misc/hash.hh"
#include <list>
#include "tgba.hh"
#include "ltlast/formula.hh"
```

Include dependency graph for tgbaexplicit.hh:



This graph shows which files directly or indirectly include this file:



## Namespaces

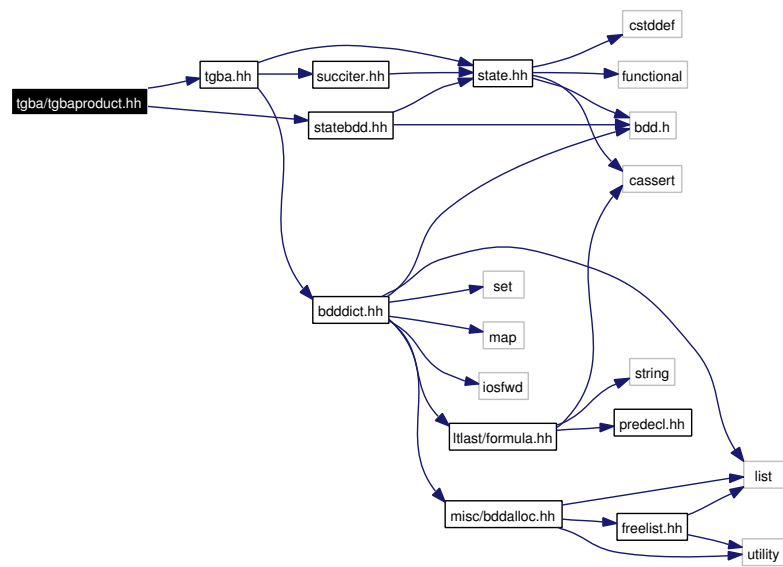
- namespace [spot](#)

## 12.79 tgba/tgbaproduct.hh File Reference

```
#include "tgba.hh"
```

```
#include "statebdd.hh"
```

Include dependency graph for tgbaproduct.hh:



## Namespaces

- namespace [spot](#)

## 12.80 tgba/tgbareduc.hh File Reference

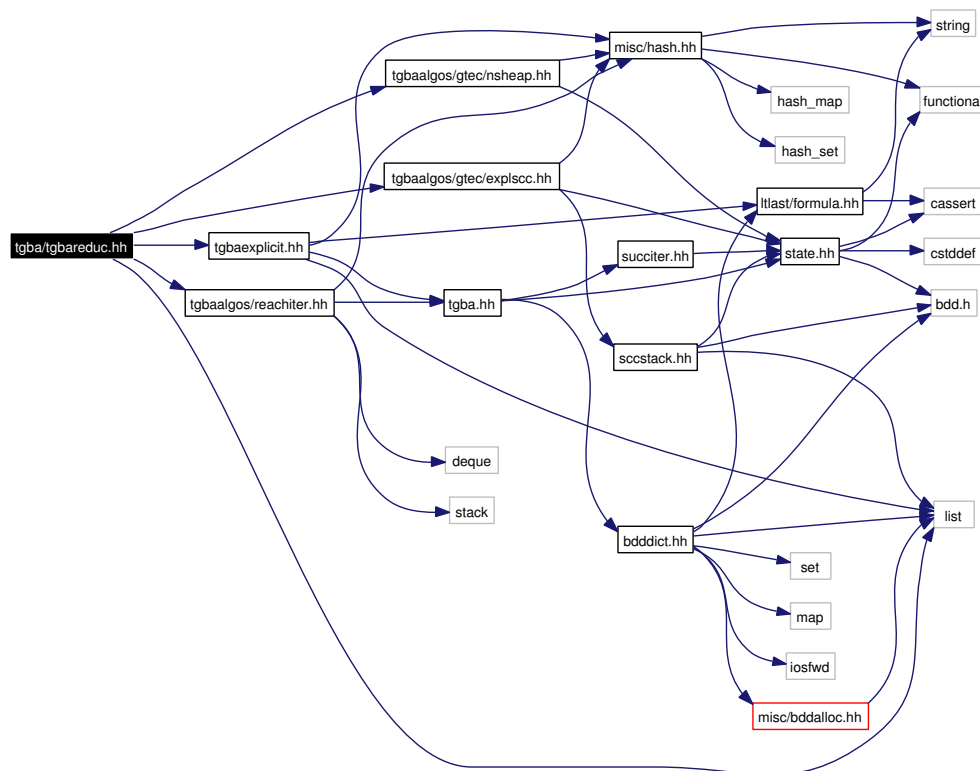
```

#include "tgbaexplicit.hh"
#include "tgbaalgos/reachiter.hh"
#include "tgbaalgos/gtec/explsccl.hh"
#include "tgbaalgos/gtec/nsheap.hh"
#include <list>

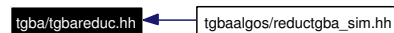
```

Include dependency graph for `tgbareduc.hh`:





This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [spot](#)

## Typedefs

- typedef Sgi::pair< const [spot::state](#) \*, const [spot::state](#) \* > [state\\_couple](#)
- typedef Sgi::vector< [state\\_couple](#) \* > [simulation\\_relation](#)

### 12.80.1 Typedef Documentation

**12.80.1.1** typedef Sgi::vector<[state\\_couple](#)\*> [spot::simulation\\_relation](#)

**12.80.1.2** typedef Sgi::pair<const [spot::state](#)\*, const [spot::state](#)\*> [spot::state\\_couple](#)

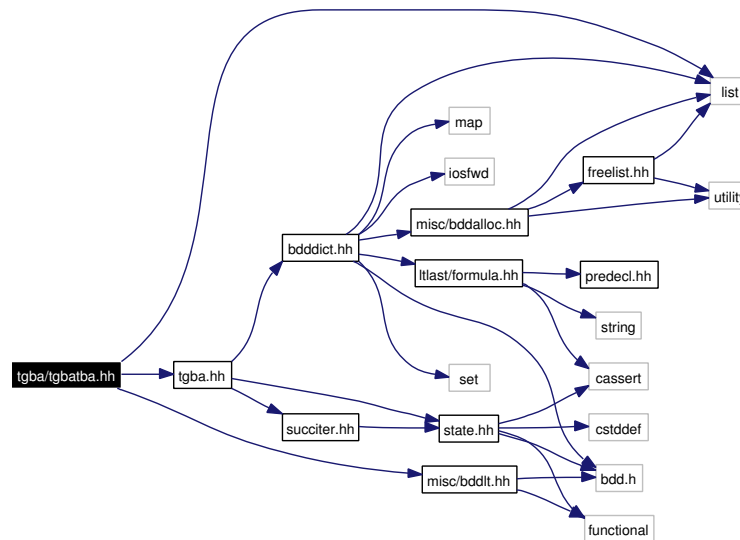
## 12.81 tgba/tgbatba.hh File Reference

```
#include <list>
```

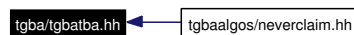
```
#include "tgba.hh"
```

```
#include "misc/bddlt.hh"
```

Include dependency graph for tgbatba.hh:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [spot](#)

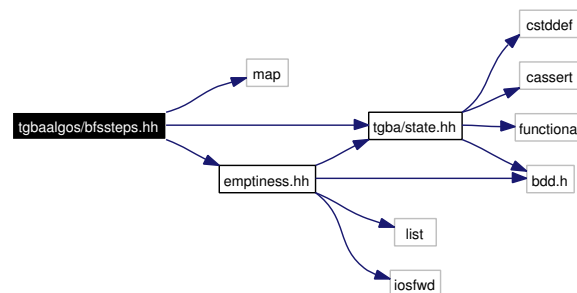
## 12.82 tgbaalgos/bfssteps.hh File Reference

```
#include <map>
```

```
#include "tgba/state.hh"
```

```
#include "emptiness.hh"
```

Include dependency graph for bfssteps.hh:



## Namespaces

- namespace [spot](#)

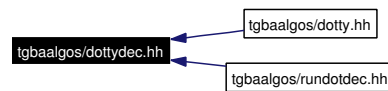
## 12.83 tgbalgorithms/dottydec.hh File Reference

```
#include <string>
```

Include dependency graph for dottydec.hh:



This graph shows which files directly or indirectly include this file:



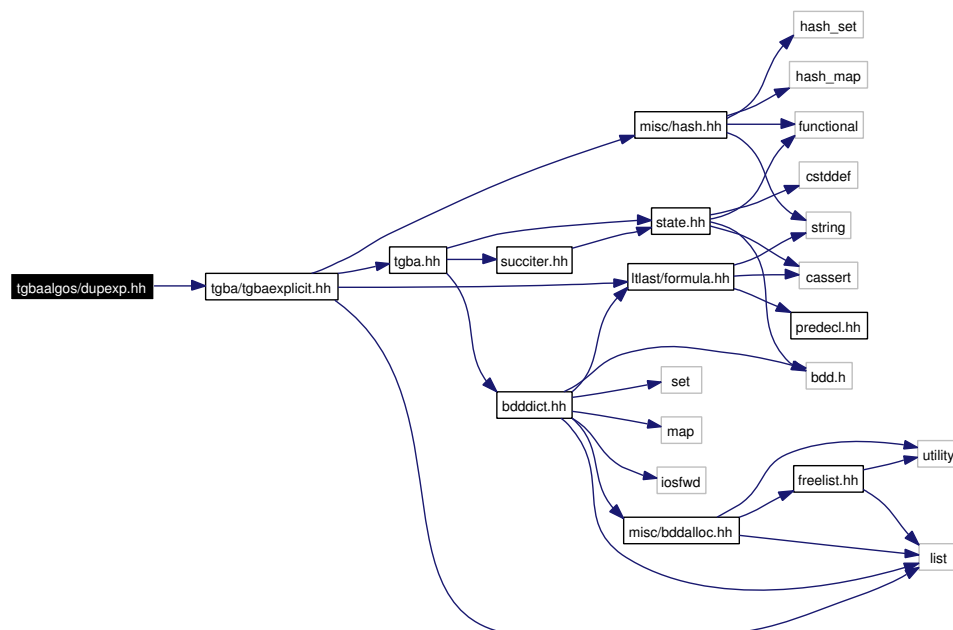
## Namespaces

- namespace [spot](#)

## 12.84 tgbalgorithms/dupeexp.hh File Reference

```
#include "tgba/tgbaexplicit.hh"
```

Include dependency graph for dupeexp.hh:



## Namespaces

- namespace [spot](#)

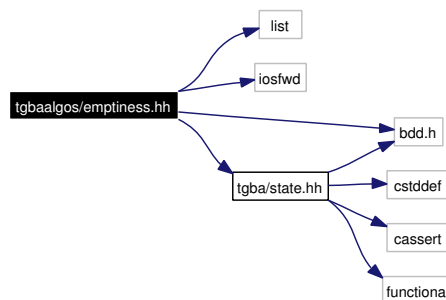
## Functions

- tgba\_explicit \* [tgba\\_dupexp\\_bfs](#) (const tgba \*aut)  
*Build an explicit automata from all states of aut, numbering states in bread first order as they are processed.*
- tgba\_explicit \* [tgba\\_dupexp\\_dfs](#) (const tgba \*aut)  
*Build an explicit automata from all states of aut, numbering states in depth first order as they are processed.*

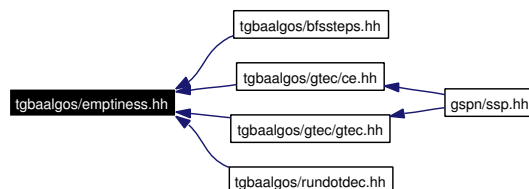
## 12.85 tgbaalgos/emptiness.hh File Reference

```
#include <list>
#include <iosfwd>
#include <bdd.h>
#include "tgba/state.hh"
```

Include dependency graph for emptiness.hh:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [spot](#)

## Functions

- `std::ostream & print_tgba_run` (`std::ostream &os`, `const tgba *a`, `const tgba_run *run`)  
*Display a `tgba_run`.*
- `tgba * tgba_run_to_tgba` (`const tgba *a`, `const tgba_run *run`)  
*Return an explicit `tgba` corresponding to run (i.e. comparable states are merged).*

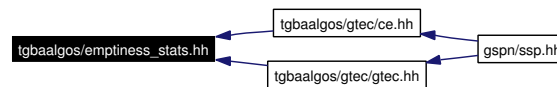
## 12.86 tgbaalgos/emptiness\_stats.hh File Reference

```
#include <cassert>
```

Include dependency graph for `emptiness_stats.hh`:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace `spot`

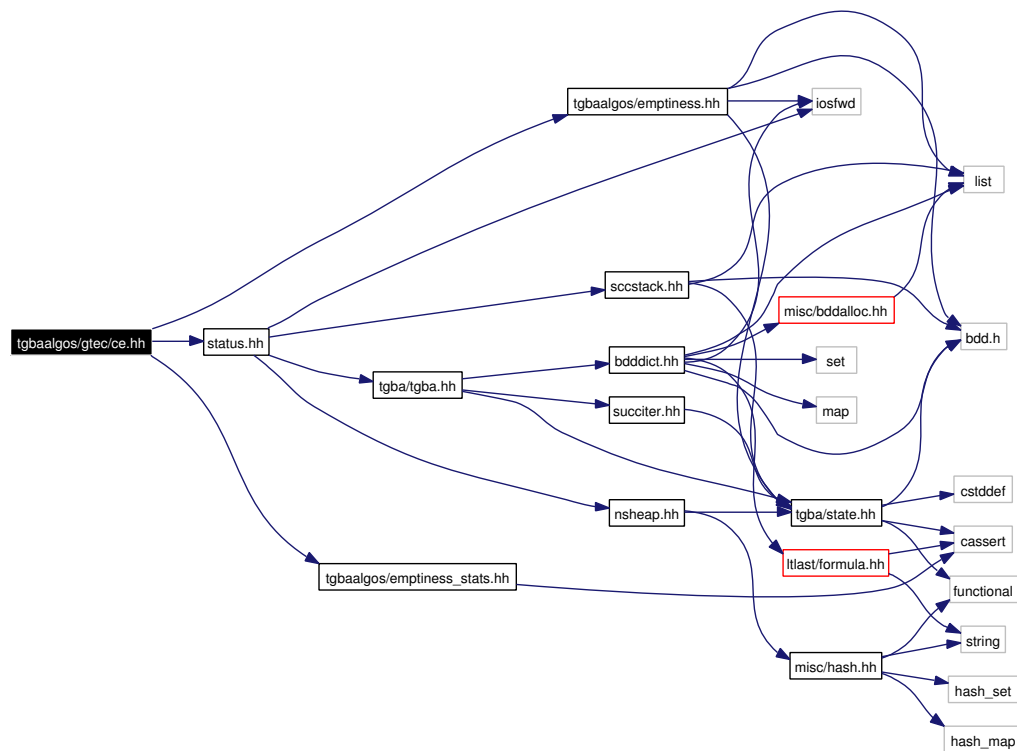
## 12.87 tgbaalgos/gtec/ce.hh File Reference

```
#include "status.hh"
```

```
#include "tgbaalgos/emptiness.hh"
```

```
#include "tgbaalgos/emptiness_stats.hh"
```

Include dependency graph for `ce.hh`:



This graph shows which files directly or indirectly include this file:



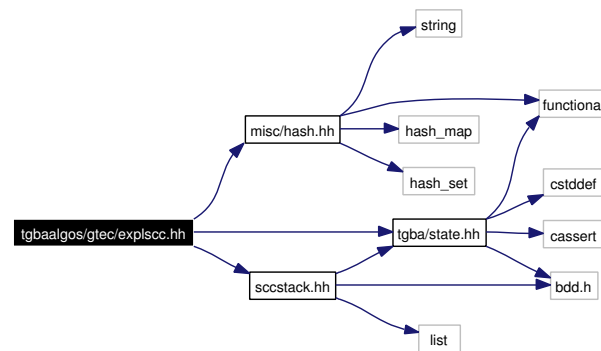
## Namespaces

- namespace [spot](#)

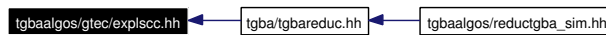
## 12.88 tgbalgorithms/gtec/explsc.h File Reference

```
#include "misc/hash.h"
#include "tgba/state.h"
#include "sccstack.h"
```

Include dependency graph for explsc.h:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [spot](#)

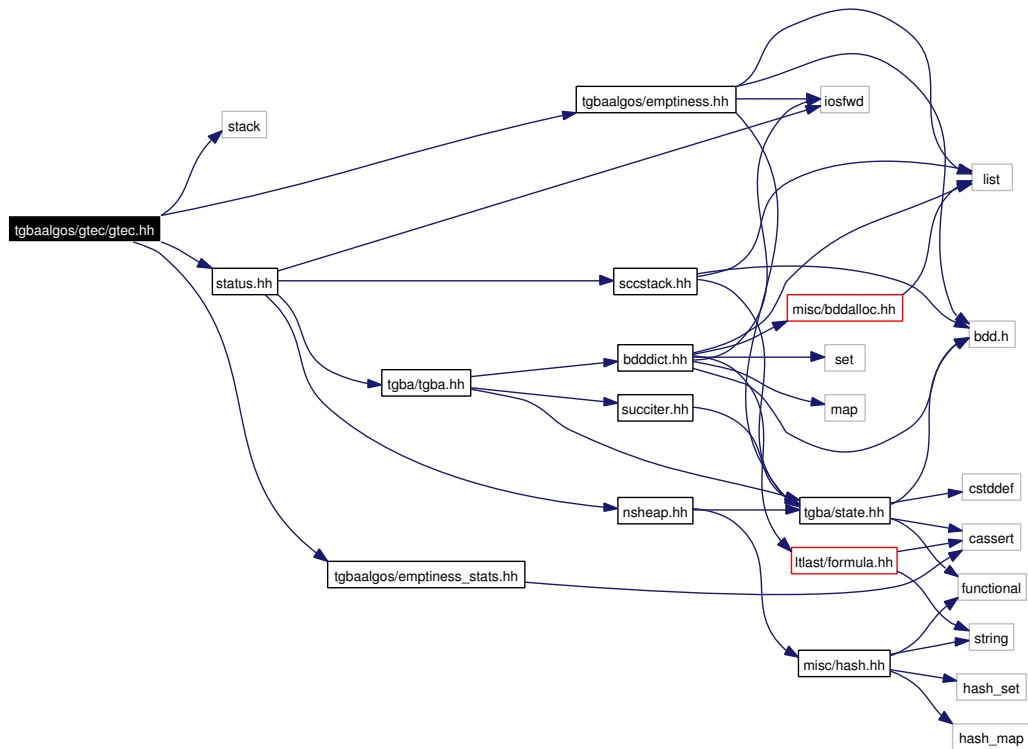
## 12.89 tgbaalgos/gtec/gtec.hh File Reference

```

#include <stack>
#include "status.hh"
#include "tgbaalgos/emptiness.hh"
#include "tgbaalgos/emptiness_stats.hh"

```

Include dependency graph for `gtec.hh`:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [spot](#)

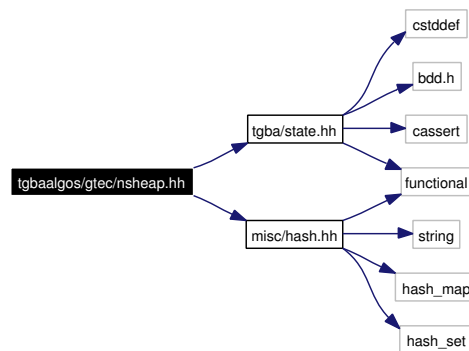
## 12.90 tgbaalgos/gtec/nsheap.hh File Reference

```
#include "tgba/state.hh"
```

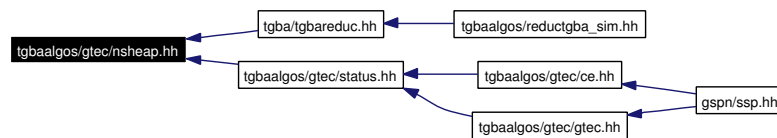
```
#include "misc/hash.hh"
```

Include dependency graph for nsheap.hh:





This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [spot](#)

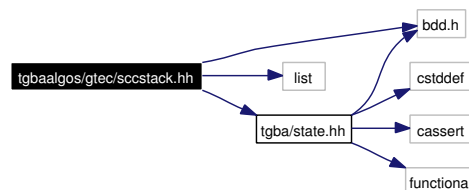
## 12.91 tgbaalgos/gtec/sccstack.hh File Reference

```
#include <bdd.h>
```

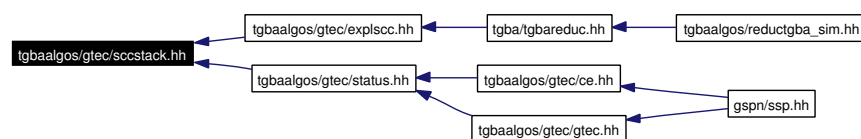
```
#include <list>
```

```
#include <tgba/state.hh>
```

Include dependency graph for sccstack.hh:



This graph shows which files directly or indirectly include this file:



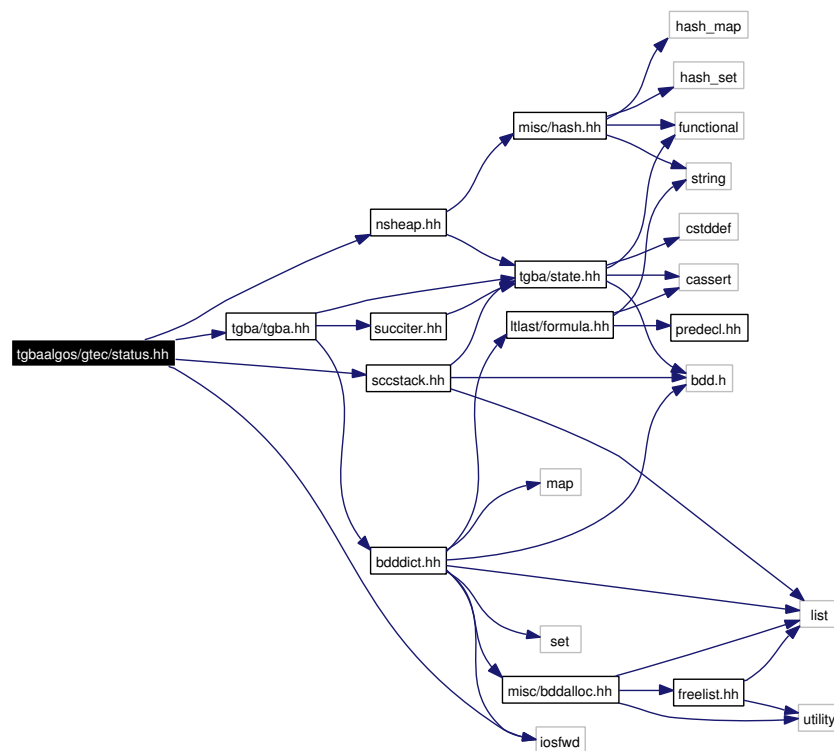
## Namespaces

- namespace [spot](#)

## 12.92 tgbaalgos/gtec/status.hh File Reference

```
#include "sccstack.hh"
#include "nsheap.hh"
#include "tgba/tgba.hh"
#include <iosfwd>
```

Include dependency graph for status.hh:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [spot](#)

## 12.93 tgbaalgos/gv04.hh File Reference

### Namespaces

- namespace [spot](#)

### Functions

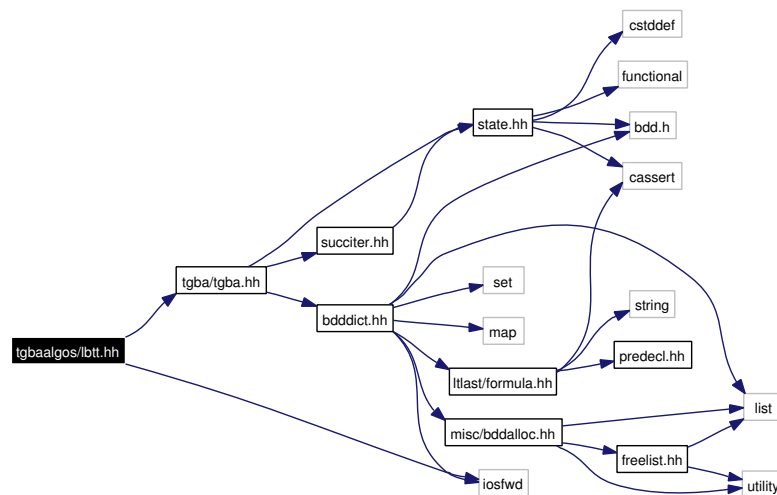
- emptiness\_check \* [explicit\\_gv04\\_check](#) (const tgba \*a)  
*Emptiness check based on Geldenhuys and Valmari's TACAS'04 paper.*

## 12.94 tgbaalgos/lbtt.hh File Reference

```
#include "tgba/tgba.hh"
```

```
#include <iosfwd>
```

Include dependency graph for lbtt.hh:



### Namespaces

- namespace [spot](#)

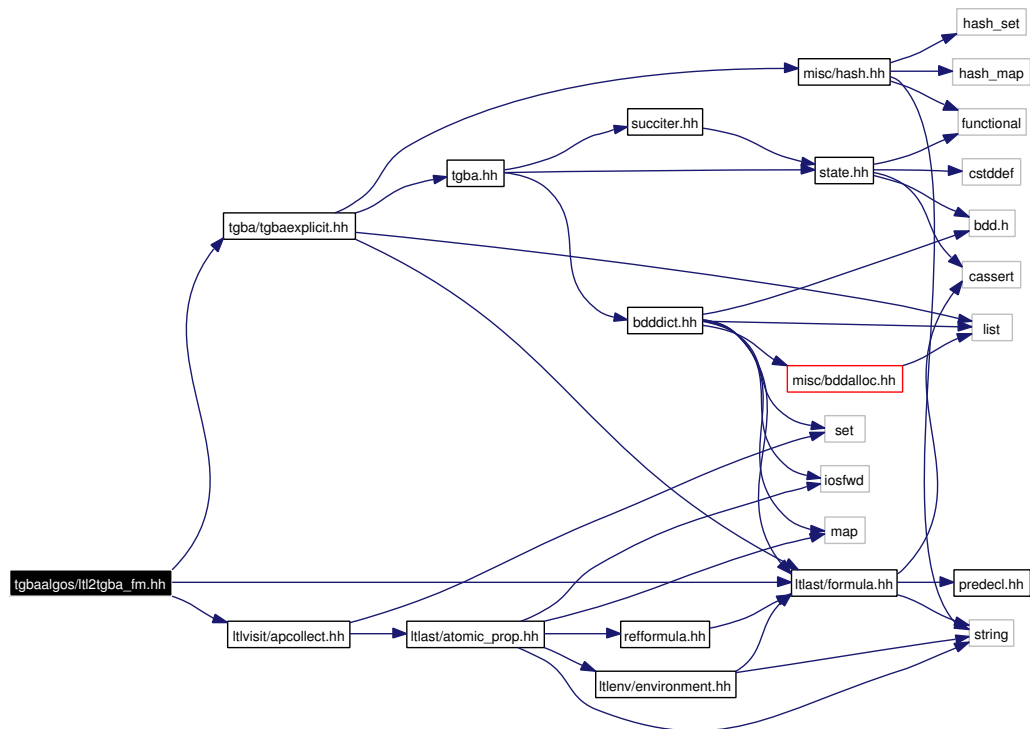
### Functions

- std::ostream & [lbtt\\_reachable](#) (std::ostream &os, const tgba \*g)  
*Print reachable states in LBTT format.*

## 12.95 tgbaalgos/ltl2tgba\_fm.hh File Reference

```
#include "ltlast/formula.hh"
```

```
#include "tgba/tgbaexplicit.hh"
#include "ltlvisit/apcollect.hh"
Include dependency graph for ltl2tgba_fm.hh:
```



## Namespaces

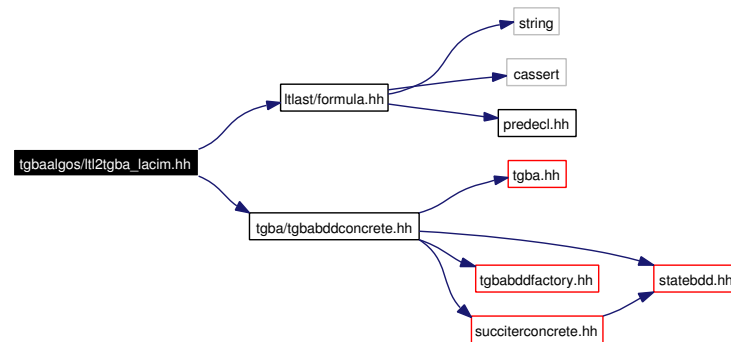
- namespace **spot**

## Functions

- `rgba_explicit * ltl_to_rgba_fm` (const ltl::formula \*f, bdd\_dict \*dict, bool exprop=false, bool symb\_merge=true, bool branching\_postponement=false, bool fair\_loop\_approx=false, const ltl::atomic\_prop\_set \*unobs=0)
- Build a `spot::rgba_explicit` from an LTL formula.*

## 12.96 tgbaalgos/ltl2tgba\_lacim.hh File Reference

```
#include "ltlast/formula.hh"
#include "tgba/tgababddconcrete.hh"
Include dependency graph for lt2tgba_lacim.hh:
```



## Namespaces

- namespace [spot](#)

## Functions

- `tgba_bdd_concrete * ltl\_to\_tgba\_lacim` (const ltl::formula \*f, bdd\_dict \*dict)  
*Build a [spot::tgba\\_bdd\\_concrete](#) from an LTL formula.*

## 12.97 tgbaalgos/magic.hh File Reference

```
#include <cstdint>
```

Include dependency graph for magic.hh:



## Namespaces

- namespace [spot](#)

## Functions

- `emptiness_check * explicit\_magic\_search` (const tgba \*a)  
*Returns an emptiness checker on the [spot::tgba](#) automaton a.*
- `emptiness_check * bit\_state\_hashing\_magic\_search` (const tgba \*a, size\_t size)  
*Returns an emptiness checker on the [spot::tgba](#) automaton a.*

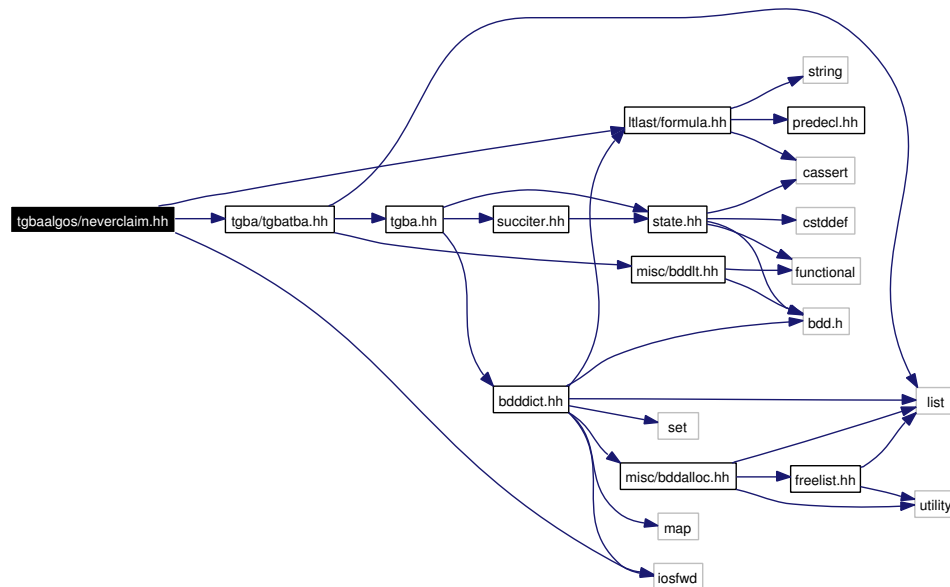
## 12.98 tgbaalgos/neverclaim.hh File Reference

```
#include <iosfwd>
```

```
#include "ltlast/formula.hh"
```

```
#include "tgba/tgbatba.hh"
```

Include dependency graph for neverclaim.hh:



## Namespaces

- namespace [spot](#)

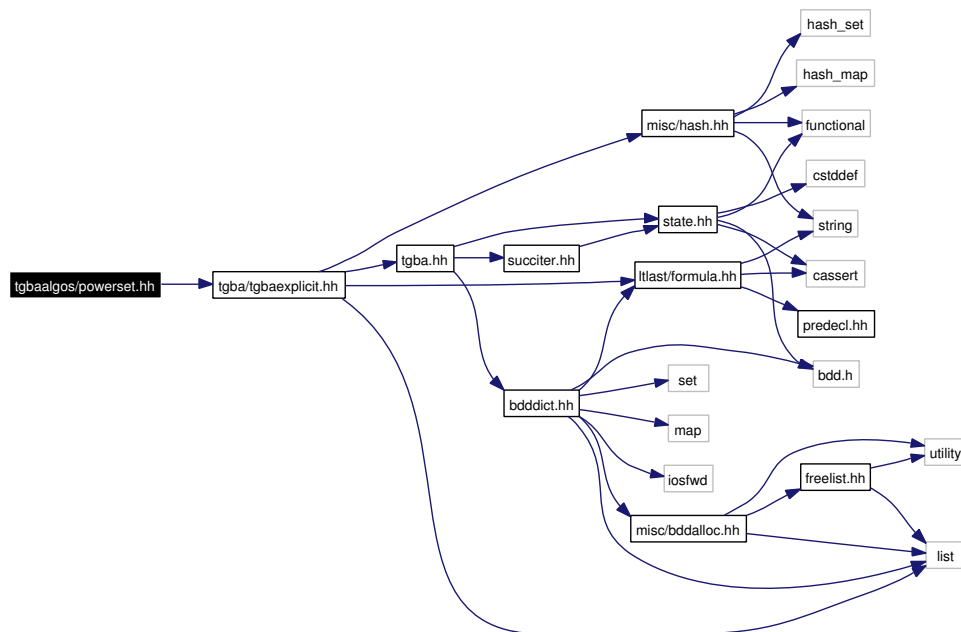
## Functions

- `std::ostream & never\_claim\_reachable (std::ostream &os, const tgba_sba_proxy *g, const ltl::formula *f=0)`  
*Print reachable states in Spin never claim format.*

## 12.99 tgbaalgos/powerset.hh File Reference

```
#include "tgba/tgbaexplicit.hh"
```

Include dependency graph for powerset.hh:



## Namespaces

- namespace [spot](#)

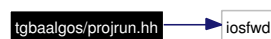
## Functions

- tgba\_explicit \* [tgba\\_powerset](#) (const tgba \*aut)  
*Build a deterministic automaton, ignoring acceptance conditions.*

## 12.100 tgbaalgos/projrun.hh File Reference

```
#include <iosfwd>
```

Include dependency graph for projrun.hh:



## Namespaces

- namespace [spot](#)

## Functions

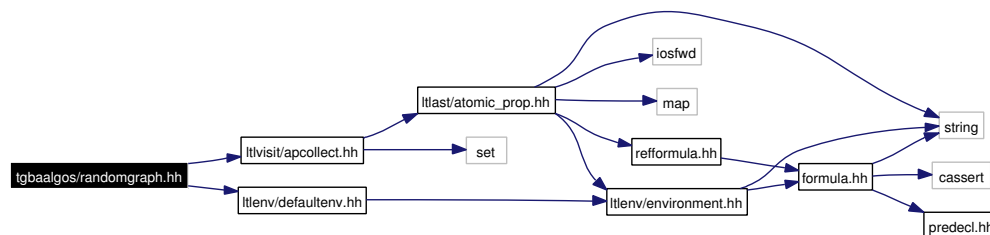
- tgba\_run \* [project\\_tgba\\_run](#) (const tgba \*a\_run, const tgba \*a\_proj, const tgba\_run \*run)  
*Project a [tgba\\_run](#) on a tgba.*

## 12.101 tgbaalgos/randomgraph.hh File Reference

```
#include "ltlvisit/apcollect.hh"
```

```
#include "ltlenv/defaultenv.hh"
```

Include dependency graph for randomgraph.hh:



### Namespaces

- namespace [spot](#)

### Functions

- tgba \* [random\\_graph](#) (int n, float d, const [ltl::atomic\\_prop\\_set](#) \*ap, bdd\_dict \*dict, int n\_acc=0, float a=0.1, float t=0.5, ltl::environment \*env=&ltl::default\_environment::instance())  
Construct a tgba randomly.

## 12.102 tgbaalgos/reducerun.hh File Reference

### Namespaces

- namespace [spot](#)

### Functions

- tgba\_run \* [reduce\\_run](#) (const tgba \*a, const tgba\_run \*org)  
Reduce an accepting run.

## 12.103 tgbaalgos/reductgba\_sim.hh File Reference

```
#include "tgba/tgbareduc.hh"
```

```
#include "tgbaalgos/reachiter.hh"
```

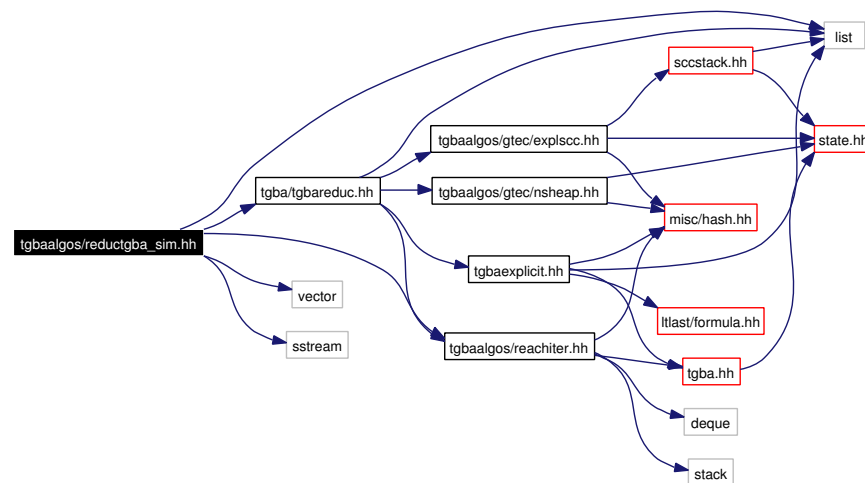
```
#include <vector>
```

```
#include <list>
```

```
#include <sstream>
```

Include dependency graph for reductgba\_sim.hh:





## Namespaces

- namespace [spot](#)

## Typedefs

- typedef Sgi::vector< spoiler\_node \* > [sn\\_v](#)
- typedef Sgi::vector< duplicator\_node \* > [dn\\_v](#)
- typedef Sgi::vector< const state \* > [s\\_v](#)

## Enumerations

- enum [reduce\\_tgba\\_options](#) {  
[Reduce\\_None](#) = 0, [Reduce\\_quotient\\_Dir\\_Sim](#) = 1, [Reduce\\_transition\\_Dir\\_Sim](#) = 2, [Reduce\\_](#)  
[quotient\\_Del\\_Sim](#) = 4,  
[Reduce\\_transition\\_Del\\_Sim](#) = 8, [Reduce\\_Scc](#) = 16, [Reduce\\_All](#) = -1U }  
*Options for reduce.*

## Functions

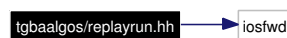
- tgba \* [reduc\\_tgba\\_sim](#) (const tgba \*a, int opt=[Reduce\\_All](#))  
*Remove some node of the automata using a simulation relation.*
- direct\_simulation\_relation \* [get\\_direct\\_relation\\_simulation](#) (const tgba \*a, std::ostream &os, int opt=-1)  
*Compute a direct simulation relation on state of tgba f.*
- delayed\_simulation\_relation \* [get\\_delayed\\_relation\\_simulation](#) (const tgba \*a, std::ostream &os, int opt=-1)  
*Compute a delayed simulation relation on state of tgba f.*

- void [free\\_relation\\_simulation](#) (direct\_simulation\_relation \*rel)  
*To free a simulation relation.*
- void [free\\_relation\\_simulation](#) (delayed\_simulation\_relation \*rel)  
*To free a simulation relation.*

## 12.104 tgbaalgos/replayrun.hh File Reference

```
#include <iosfwd>
```

Include dependency graph for replayrun.hh:



### Namespaces

- namespace [spot](#)

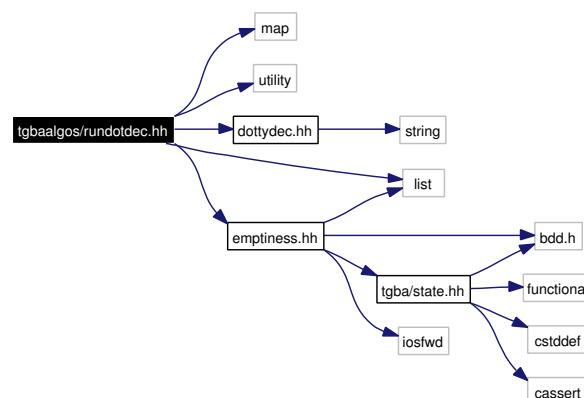
### Functions

- bool [replay\\_tgba\\_run](#) (std::ostream &os, const tgba \*a, const tgba\_run \*run, bool debug=false)  
*Replay a [tgba\\_run](#) on a [tgba](#).*

## 12.105 tgbaalgos/rundotdec.hh File Reference

```
#include <map>
#include <utility>
#include <list>
#include "dottydec.hh"
#include "emptiness.hh"
```

Include dependency graph for rundotdec.hh:



## Namespaces

- namespace [spot](#)

## 12.106 tgbaalgos/se05.hh File Reference

```
#include <cstdint>
```

Include dependency graph for se05.hh:



## Namespaces

- namespace [spot](#)

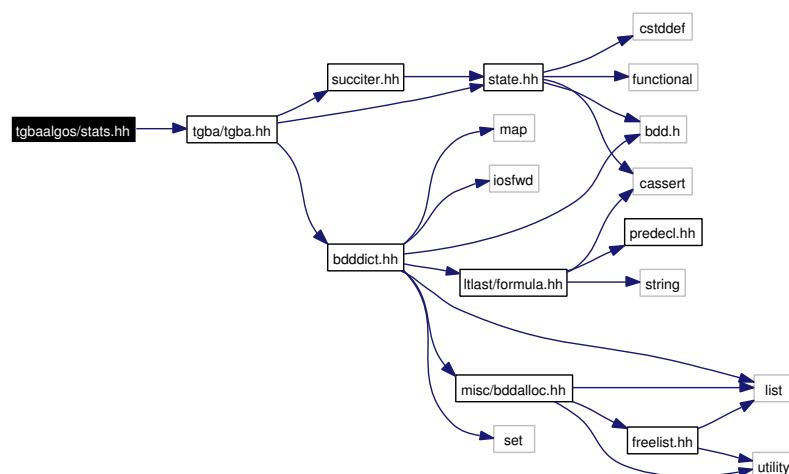
## Functions

- emptiness\_check \* [explicit\\_se05\\_search](#) (const tgba \*a)  
Returns an emptiness check on the [spot::tgba](#) automaton a.
- emptiness\_check \* [bit\\_state\\_hashing\\_se05\\_search](#) (const tgba \*a, size\_t size)  
Returns an emptiness checker on the [spot::tgba](#) automaton a.

## 12.107 tgbaalgos/stats.hh File Reference

```
#include "tgba/tgba.hh"
```

Include dependency graph for stats.hh:



## Namespaces

- namespace [spot](#)

## Functions

- tgba\_statistics [stats\\_reachable](#) (const tgba \*g)  
*Compute statistics for an automaton.*

## 12.108 tgbaalgos/tau03.hh File Reference

### Namespaces

- namespace [spot](#)

### Functions

- emptiness\_check \* [explicit\\_tau03\\_search](#) (const tgba \*a)  
*Returns an emptiness checker on the [spot::tgba](#) automaton a.*

## 12.109 tgbaalgos/tau03opt.hh File Reference

### Namespaces

- namespace [spot](#)

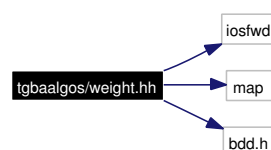
### Functions

- emptiness\_check \* [explicit\\_tau03\\_opt\\_search](#) (const tgba \*a)  
*Returns an emptiness checker on the [spot::tgba](#) automaton a.*

## 12.110 tgbaalgos/weight.hh File Reference

```
#include <iosfwd>
#include <map>
#include <bdd.h>
```

Include dependency graph for weight.hh:



## Namespaces

- namespace [spot](#)

## Index

- ~atomic\_prop
  - spot::ltl::atomic\_prop, [82](#)
- ~bdd\_dict
  - spot::bdd\_dict, [95](#)
- ~bfs\_steps
  - spot::bfs\_steps, [104](#)
- ~binop
  - spot::ltl::binop, [110](#)
- ~clone\_visitor
  - spot::ltl::clone\_visitor, [114](#)
- ~connected\_component\_hash\_set
  - spot::connected\_component\_hash\_set, [117](#)
- ~connected\_component\_hash\_set\_factory
  - spot::connected\_component\_hash\_set\_factory, [119](#)
- ~constant
  - spot::ltl::constant, [123](#)
- ~couvreur99\_check
  - spot::couvreur99\_check, [127](#)
- ~couvreur99\_check\_shy
  - spot::couvreur99\_check\_shy, [136](#)
- ~couvreur99\_check\_status
  - spot::couvreur99\_check\_status, [141](#)
- ~declarative\_environment
  - spot::ltl::declarative\_environment, [143](#)
- ~default\_environment
  - spot::ltl::default\_environment, [145](#)
- ~dotty\_decorator
  - spot::dotty\_decorator, [147](#)
- ~duplicator\_node
  - spot::duplicator\_node, [151](#)
- ~duplicator\_node\_delayed
  - spot::duplicator\_node\_delayed, [155](#)
- ~emptiness\_check
  - spot::emptiness\_check, [162](#)
- ~environment
  - spot::ltl::environment, [165](#)
- ~evtgba
  - spot::evtgba, [167](#)
- ~evtgba\_explicit
  - spot::evtgba\_explicit, [170](#)
- ~evtgba\_iterator
  - spot::evtgba\_iterator, [174](#)
- ~evtgba\_product
  - spot::evtgba\_product, [176](#)
- ~evtgba\_reachable\_iterator
  - spot::evtgba\_reachable\_iterator, [180](#)
- ~explicit\_connected\_component
  - spot::explicit\_connected\_component, [190](#)
- ~explicit\_connected\_component\_factory
  - spot::explicit\_connected\_component\_factory, [191](#)
- ~formula
  - spot::ltl::formula, [193](#)
- ~free\_list
  - spot::free\_list, [198](#)
- ~gspn\_interface
  - spot::gspn\_interface, [201](#)
- ~gspn\_ssp\_interface
  - spot::gspn\_ssp\_interface, [203](#)
- ~loopless\_modular\_mixed\_radix\_gray\_code
  - spot::loopless\_modular\_mixed\_radix\_gray\_code, [207](#)
- ~multop
  - spot::ltl::multop, [217](#)
- ~numbered\_state\_heap
  - spot::numbered\_state\_heap, [221](#)
- ~numbered\_state\_heap\_const\_iterator
  - spot::numbered\_state\_heap\_const\_iterator, [223](#)
- ~numbered\_state\_heap\_factory
  - spot::numbered\_state\_heap\_factory, [224](#)
- ~numbered\_state\_heap\_hash\_map
  - spot::numbered\_state\_heap\_hash\_map, [227](#)
- ~numbered\_state\_heap\_hash\_map\_factory
  - spot::numbered\_state\_heap\_hash\_map\_factory, [229](#)
- ~parity\_game\_graph
  - spot::parity\_game\_graph, [232](#)
- ~parity\_game\_graph\_delayed
  - spot::parity\_game\_graph\_delayed, [238](#)
- ~parity\_game\_graph\_direct
  - spot::parity\_game\_graph\_direct, [245](#)
- ~postfix\_visitor
  - spot::ltl::postfix\_visitor, [250](#)
- ~random\_ltl
  - spot::ltl::random\_ltl, [253](#)
- ~ref\_formula
  - spot::ltl::ref\_formula, [258](#)
- ~rsymbol
  - spot::rsymbol, [260](#)
- ~simplify\_f\_g\_visitor
  - spot::ltl::simplify\_f\_g\_visitor, [266](#)
- ~spoiler\_node
  - spot::spoiler\_node, [269](#)
- ~spoiler\_node\_delayed
  - spot::spoiler\_node\_delayed, [272](#)
- ~state
  - spot::state, [277](#)
- ~state\_evtgba\_explicit

- spot::state\_evtgba\_explicit, 282
- ~state\_explicit
  - spot::state\_explicit, 284
- ~state\_product
  - spot::state\_product, 287
- ~symbol
  - spot::symbol, 292
- ~tgba
  - spot::tgba, 295
- ~tgba\_bdd\_concrete
  - spot::tgba\_bdd\_concrete, 302
- ~tgba\_bdd\_concrete\_factory
  - spot::tgba\_bdd\_concrete\_factory, 307
- ~tgba\_explicit
  - spot::tgba\_explicit, 320
- ~tgba\_explicit\_succ\_iterator
  - spot::tgba\_explicit\_succ\_iterator, 327
- ~tgba\_product
  - spot::tgba\_product, 332
- ~tgba\_reachable\_iterator
  - spot::tgba\_reachable\_iterator, 337
- ~tgba\_reduc
  - spot::tgba\_reduc, 353
- ~tgba\_run
  - spot::tgba\_run, 362
- ~tgba\_run\_dotty\_decorator
  - spot::tgba\_run\_dotty\_decorator, 365
- ~tgba\_succ\_iterator
  - spot::tgba\_succ\_iterator, 375
- ~tgba\_succ\_iterator\_concrete
  - spot::tgba\_succ\_iterator\_concrete, 378
- ~tgba\_succ\_iterator\_product
  - spot::tgba\_succ\_iterator\_product, 383
- ~tgba\_tba\_proxy
  - spot::tgba\_tba\_proxy, 388
- ~unabbreviate\_logic\_visitor
  - spot::ltl::unabbreviate\_logic\_visitor, 398
- ~unabbreviate\_ltl\_visitor
  - spot::ltl::unabbreviate\_ltl\_visitor, 401
- ~unop
  - spot::ltl::unop, 406
- a\_
  - spot::bfs\_steps, 105
  - spot::couvereur99\_check, 128
  - spot::couvereur99\_check\_result, 132
  - spot::couvereur99\_check\_shy, 137
  - spot::emptiness\_check, 162
  - spot::emptiness\_check\_result, 164
  - spot::loopless\_modular\_mixed\_radix\_gray\_
    - code, 208
  - spot::tgba\_tba\_proxy, 391
- a\_first
  - spot::loopless\_modular\_mixed\_radix\_gray\_
    - code, 208
- a\_last
  - spot::loopless\_modular\_mixed\_radix\_gray\_
    - code, 208
- a\_next
  - spot::loopless\_modular\_mixed\_radix\_gray\_
    - code, 208
- acc
  - spot::couvereur99\_check\_shy::successor, 138
  - spot::tgba\_run::step, 363
- acc\_
  - spot::duplicator\_node, 152
  - spot::duplicator\_node\_delayed, 157
  - spot::tgba\_bdd\_concrete\_factory, 309
  - spot::tgba\_reduc, 361
- acc\_cycle\_
  - spot::tgba\_sba\_proxy, 373
  - spot::tgba\_tba\_proxy, 391
- acc\_formula\_map
  - spot::bdd\_dict, 98
- acc\_map
  - spot::bdd\_dict, 98
- acc\_map\_
  - spot::tgba\_bdd\_concrete\_factory, 307
- acc\_set
  - spot::tgba\_bdd\_core\_data, 312
- acc\_set\_
  - spot::evtgba\_explicit, 172
- accept
  - spot::ltl::atomic\_prop, 82
  - spot::ltl::binop, 110
  - spot::ltl::constant, 123
  - spot::ltl::formula, 193
  - spot::ltl::multop, 217
  - spot::ltl::ref\_formula, 258
  - spot::ltl::unop, 407
- acceptance\_condition\_visited\_
  - spot::spoiler\_node\_delayed, 274
- acceptance\_conditions
  - spot::evtgba\_explicit::transition, 173
  - spot::tgba\_bdd\_core\_data, 312
  - spot::tgba\_explicit::transition, 325
- accepting\_cycle
  - spot::couvereur99\_check\_result, 131
- accepting\_run
  - spot::couvereur99\_check\_result, 131
  - spot::emptiness\_check\_result, 164
- acss\_states
  - spot::acss\_statistics, 76
  - spot::couvereur99\_check\_result, 131
- add\_acceptance\_condition
  - spot::tgba\_explicit, 320
  - spot::tgba\_reduc, 353

- add\_acceptance\_conditions
  - spot::tgba\_explicit, 320
  - spot::tgba\_reduc, 353
- add\_condition
  - spot::tgba\_explicit, 320
  - spot::tgba\_reduc, 354
- add\_conditions
  - spot::tgba\_explicit, 320
  - spot::tgba\_reduc, 354
- add\_duplicator\_node\_delayed
  - spot::parity\_game\_graph\_delayed, 238
- add\_pred
  - spot::duplicator\_node, 151
  - spot::duplicator\_node\_delayed, 155
  - spot::spoiler\_node, 269
  - spot::spoiler\_node\_delayed, 273
- add\_spoiler\_node\_delayed
  - spot::parity\_game\_graph\_delayed, 238
- add\_state
  - spot::evtgba\_reachable\_iterator, 180
  - spot::evtgba\_reachable\_iterator\_breadth\_first, 183
  - spot::evtgba\_reachable\_iterator\_depth\_first, 187
  - spot::parity\_game\_graph, 233
  - spot::parity\_game\_graph\_delayed, 239
  - spot::parity\_game\_graph\_direct, 245
  - spot::tgba\_explicit, 320
  - spot::tgba\_reachable\_iterator, 337
  - spot::tgba\_reachable\_iterator\_breadth\_first, 341
  - spot::tgba\_reachable\_iterator\_depth\_first, 345
  - spot::tgba\_reduc, 354
- add\_succ
  - spot::duplicator\_node, 151
  - spot::duplicator\_node\_delayed, 155
  - spot::spoiler\_node, 269
  - spot::spoiler\_node\_delayed, 273
- add\_transition
  - spot::evtgba\_explicit, 171
- Algorithm patterns, 36
- Algorithms for LTL formulae, 19
- all\_acc\_
  - spot::evtgba\_product, 177
- all\_acceptance\_conditions
  - spot::evtgba, 167
  - spot::evtgba\_explicit, 171
  - spot::evtgba\_product, 177
  - spot::tgba, 296
  - spot::tgba\_bdd\_concrete, 302
  - spot::tgba\_bdd\_core\_data, 313
  - spot::tgba\_explicit, 320
  - spot::tgba\_product, 332
  - spot::tgba\_reduc, 354
  - spot::tgba\_sba\_proxy, 370
  - spot::tgba\_tba\_proxy, 388
- all\_acceptance\_conditions\_
  - spot::tgba\_explicit, 324
  - spot::tgba\_explicit\_succ\_iterator, 328
  - spot::tgba\_product, 335
  - spot::tgba\_reduc, 361
- all\_acceptance\_conditions\_computed\_
  - spot::tgba\_explicit, 324
  - spot::tgba\_reduc, 361
- allocate\_variables
  - spot::bdd\_allocator, 89
  - spot::bdd\_dict, 96
- alphabet
  - spot::evtgba, 167
  - spot::evtgba\_explicit, 171
  - spot::evtgba\_product, 177
- alphabet\_
  - spot::evtgba\_explicit, 172
  - spot::evtgba\_product, 177
- And
  - spot::ltl::multop, 217
- annon\_free\_list
  - spot::bdd\_dict::annon\_free\_list, 101
- ap
  - spot::ltl::random\_ltl, 253
- ap\_
  - spot::ltl::random\_ltl, 254
- arc
  - spot::couvreur99\_check\_shy, 137
- ars\_cycle\_states
  - spot::ars\_statistics, 77
  - spot::couvreur99\_check\_result, 131
- ars\_prefix\_states
  - spot::ars\_statistics, 77
  - spot::couvreur99\_check\_result, 131
- ars\_statistics
  - spot::ars\_statistics, 77
- as\_bdd
  - spot::state\_bdd, 280
- assert\_emptiness
  - spot::bdd\_dict, 96
- atomic\_prop
  - spot::ltl::atomic\_prop, 82
- atomic\_prop\_collect
  - ltl\_misc, 25
- atomic\_prop\_set
  - ltl\_misc, 25
- aut
  - spot::couvreur99\_check\_status, 141
- automata\_
  - spot::evtgba\_reachable\_iterator, 181
  - spot::evtgba\_reachable\_iterator\_breadth\_first, 184



- spot::evtgba\_reachable\_iterator\_depth\_first, 188
- spot::parity\_game\_graph, 234
- spot::parity\_game\_graph\_delayed, 240
- spot::parity\_game\_graph\_direct, 246
- spot::tgba\_reachable\_iterator, 338
- spot::tgba\_reachable\_iterator\_breadth\_first, 343
- spot::tgba\_reachable\_iterator\_depth\_first, 346
- spot::tgba\_reduc, 361
- automaton
  - spot::couvereur99\_check, 127
  - spot::couvereur99\_check\_result, 132
  - spot::couvereur99\_check\_shy, 136
  - spot::emptiness\_check, 162
  - spot::emptiness\_check\_result, 164
  - spot::gspn\_interface, 202
  - spot::gspn\_ssp\_interface, 204
- barand
  - spot::barand, 85
- basic\_reduce
  - ltl\_rewriting, 23
- bdd\_allocator
  - spot::bdd\_allocator, 89
- bdd\_dict
  - spot::bdd\_dict, 95
- bdd\_format\_accset
  - bddprint.hh, 457
  - spot, 67
- bdd\_format\_formula
  - bddprint.hh, 458
  - spot, 67
- bdd\_format\_sat
  - bddprint.hh, 458
  - spot, 67
- bdd\_format\_set
  - bddprint.hh, 458
  - spot, 67
- bdd\_print\_acc
  - bddprint.hh, 458
  - spot, 67
- bdd\_print\_accset
  - bddprint.hh, 459
  - spot, 68
- bdd\_print\_dot
  - bddprint.hh, 459
  - spot, 68
- bdd\_print\_formula
  - bddprint.hh, 459
  - spot, 68
- bdd\_print\_sat
  - bddprint.hh, 459
  - spot, 68
- bdd\_print\_set
  - bddprint.hh, 459
  - spot, 69
- bdd\_print\_table
  - bddprint.hh, 460
  - spot, 69
- bdd\_to\_formula
  - formula2bdd.hh, 461
  - spot, 69
- bdd\_v
  - spot::parity\_game\_graph\_delayed, 238
- bddprint.hh
  - bdd\_format\_accset, 457
  - bdd\_format\_formula, 458
  - bdd\_format\_sat, 458
  - bdd\_format\_set, 458
  - bdd\_print\_acc, 458
  - bdd\_print\_accset, 459
  - bdd\_print\_dot, 459
  - bdd\_print\_formula, 459
  - bdd\_print\_sat, 459
  - bdd\_print\_set, 459
  - bdd\_print\_table, 460
- begin
  - yy::Location, 206
  - yy::Stack, 276
- bfs\_steps
  - spot::bfs\_steps, 104
- binop
  - spot::ltl::binop, 110
- bit\_state\_hashing\_magic\_search
  - emptiness\_check\_algorithms, 42
- bit\_state\_hashing\_se05\_search
  - emptiness\_check\_algorithms, 43
- bmrand
  - random, 30
- build
  - spot::connected\_component\_hash\_set\_factory, 119
  - spot::explicit\_connected\_component\_factory, 191
  - spot::ltl::random\_ltl::op\_proba, 255
  - spot::numbered\_state\_heap\_factory, 224
  - spot::numbered\_state\_heap\_hash\_map\_factory, 229
- build\_graph
  - spot::parity\_game\_graph, 233
  - spot::parity\_game\_graph\_delayed, 239
  - spot::parity\_game\_graph\_direct, 245
- build\_link
  - spot::parity\_game\_graph\_direct, 245
- build\_recurse\_successor\_duplicator
  - spot::parity\_game\_graph\_delayed, 239
- build\_recurse\_successor\_spoiler

- spot::parity\_game\_graph\_delayed, 239
- builder
  - spot::ltl::random\_ltl::op\_proba, 255
- cancel
  - spot::timer\_map, 394
- check
  - spot::couvreur99\_check, 127
  - spot::couvreur99\_check\_shy, 136
  - spot::emptiness\_check, 162
- child
  - spot::ltl::unop, 407
- child\_
  - spot::ltl::unop, 408
- children\_
  - spot::ltl::multop, 219
- clear\_rem
  - spot::scc\_stack, 262
- clear\_todo
  - spot::couvreur99\_check\_shy, 136
- clone
  - ltl\_essential, 18
  - spot::state, 277
  - spot::state\_bdd, 280
  - spot::state\_evtgba\_explicit, 282
  - spot::state\_explicit, 285
  - spot::state\_product, 288
- clone\_visitor
  - spot::ltl::clone\_visitor, 114
- column
  - yy::Position, 248
- columns
  - yy::Location, 205
  - yy::Position, 248
- common.hh
  - operator<<, 413
- common\_symbol\_table
  - spot::evtgba\_product, 176
- common\_symbols\_
  - spot::evtgba\_product, 178
- compare
  - spot::duplicator\_node, 151
  - spot::duplicator\_node\_delayed, 156
  - spot::spoiler\_node, 269
  - spot::spoiler\_node\_delayed, 273
  - spot::state, 277
  - spot::state\_bdd, 280
  - spot::state\_evtgba\_explicit, 282
  - spot::state\_explicit, 285
  - spot::state\_product, 288
- complement\_all\_acceptance\_conditions
  - spot::tgba\_explicit, 320
  - spot::tgba\_reduc, 354
- compute\_scc
  - spot::tgba\_reduc, 354
- compute\_support\_conditions
  - spot::tgba, 296
  - spot::tgba\_bdd\_concrete, 302
  - spot::tgba\_explicit, 320, 321
  - spot::tgba\_product, 332
  - spot::tgba\_reduc, 354
  - spot::tgba\_sba\_proxy, 370
  - spot::tgba\_tba\_proxy, 388
- compute\_support\_variables
  - spot::tgba, 296
  - spot::tgba\_bdd\_concrete, 302
  - spot::tgba\_explicit, 321
  - spot::tgba\_product, 332
  - spot::tgba\_reduc, 354
  - spot::tgba\_sba\_proxy, 370
  - spot::tgba\_tba\_proxy, 388
- condition
  - spot::connected\_component\_hash\_set, 117
  - spot::explicit\_connected\_component, 190
  - spot::scc\_stack::connected\_component, 264
  - spot::tgba\_explicit::transition, 325
- connected\_component
  - spot::scc\_stack::connected\_component, 264
- connected\_component\_hash\_set\_factory
  - spot::connected\_component\_hash\_set\_factory, 119
- constant
  - spot::ltl::constant, 123
- ConstIterator
  - yy::Stack, 275
- constrain\_relation
  - spot::tgba\_bdd\_concrete\_factory, 307
- copy\_acceptance\_conditions\_of
  - spot::tgba\_explicit, 321
  - spot::tgba\_reduc, 355
- couvreur99\_check
  - spot::couvreur99\_check, 127
- couvreur99\_check\_result
  - spot::couvreur99\_check\_result, 131
- couvreur99\_check\_shy
  - spot::couvreur99\_check\_shy, 136
- couvreur99\_check\_ssp\_semi
  - emptiness\_check\_ssp, 17
- couvreur99\_check\_ssp\_shy
  - emptiness\_check\_ssp, 17
- couvreur99\_check\_ssp\_shy\_semi
  - emptiness\_check\_ssp, 17
- couvreur99\_check\_status
  - spot::couvreur99\_check\_status, 141
- create\_atomic\_prop
  - spot::tgba\_bdd\_concrete\_factory, 307
- create\_state
  - spot::tgba\_bdd\_concrete\_factory, 308

- create\_transition
  - spot::tgba\_explicit, 321
  - spot::tgba\_reduc, 355
- cube\_
  - spot::minato\_isop, 210
- current\_
  - spot::tgba\_succ\_iterator\_concrete, 379
- current\_acc\_
  - spot::tgba\_succ\_iterator\_concrete, 379
- current\_acceptance\_conditions
  - spot::evtgba\_iterator, 174
  - spot::tgba\_explicit\_succ\_iterator, 327
  - spot::tgba\_succ\_iterator, 375
  - spot::tgba\_succ\_iterator\_concrete, 378
  - spot::tgba\_succ\_iterator\_product, 383
- current\_cond\_
  - spot::tgba\_succ\_iterator\_product, 384
- current\_condition
  - spot::tgba\_explicit\_succ\_iterator, 327
  - spot::tgba\_succ\_iterator, 375
  - spot::tgba\_succ\_iterator\_concrete, 378
  - spot::tgba\_succ\_iterator\_product, 383
- current\_label
  - spot::evtgba\_iterator, 174
- current\_state
  - spot::evtgba\_iterator, 174
  - spot::tgba\_explicit\_succ\_iterator, 327
  - spot::tgba\_succ\_iterator, 375
  - spot::tgba\_succ\_iterator\_concrete, 379
  - spot::tgba\_succ\_iterator\_product, 383
- current\_state\_
  - spot::tgba\_succ\_iterator\_concrete, 380
- cycle
  - spot::tgba\_run, 363
- cycle\_list
  - spot::tgba\_sba\_proxy, 370
  - spot::tgba\_tba\_proxy, 388
- cycle\_seed
  - spot::couvreur99\_check\_status, 141
- cycle\_states\_
  - spot::ars\_statistics, 78
- data\_
  - spot::tgba\_bdd\_concrete, 305
  - spot::tgba\_bdd\_concrete\_factory, 309
  - spot::tgba\_succ\_iterator\_concrete, 380
- dead\_
  - spot::gspn\_interface, 202
- dec\_depth
  - spot::couvreur99\_check, 127
  - spot::couvreur99\_check\_shy, 136
  - spot::ec\_statistics, 159
- dec\_weight\_handler
  - spot::weight, 411
- declarative\_environment
  - spot::ltl::declarative\_environment, 143
- declare
  - spot::ltl::declarative\_environment, 143
- declare\_acceptance\_condition
  - spot::evtgba\_explicit, 171
  - spot::tgba\_bdd\_concrete\_factory, 308
  - spot::tgba\_bdd\_core\_data, 312
  - spot::tgba\_explicit, 321
  - spot::tgba\_reduc, 355
- declare\_atomic\_prop
  - spot::tgba\_bdd\_core\_data, 312
- declare\_now\_next
  - spot::tgba\_bdd\_core\_data, 312
- declare\_state
  - spot::evtgba\_explicit, 171
- Decorating the dot output, 41
- default\_environment
  - spot::ltl::default\_environment, 145
- del\_pred
  - spot::duplicator\_node, 151
  - spot::duplicator\_node\_delayed, 156
  - spot::spoiler\_node, 269
  - spot::spoiler\_node\_delayed, 273
- del\_succ
  - spot::duplicator\_node, 151
  - spot::duplicator\_node\_delayed, 156
  - spot::spoiler\_node, 269
  - spot::spoiler\_node\_delayed, 273
- delete\_scc
  - spot::tgba\_reduc, 355
- delete\_transitions
  - spot::tgba\_reduc, 355
- depth
  - spot::couvreur99\_check, 127
  - spot::couvreur99\_check\_shy, 136
  - spot::ec\_statistics, 159
- depth\_
  - spot::ec\_statistics, 159
- Derivable visitors, 22
- dest
  - spot::tgba\_explicit::transition, 325
- destroy
  - ltl\_essential, 18
- dict
  - spot::tgba\_bdd\_core\_data, 313
- dict\_
  - spot::bdd\_dict::annon\_free\_list, 102
  - spot::gspn\_interface, 202
  - spot::gspn\_ssp\_interface, 204
  - spot::tgba\_explicit, 324
  - spot::tgba\_product, 335
  - spot::tgba\_reduc, 361
- display\_rel\_sim

- spot::tgba\_reduc, 355
- display\_scc
  - spot::tgba\_reduc, 355
- dn\_v
  - tgba\_reduction, 38
- doit
  - spot::ltl::postfix\_visitor, 250
- doit\_default
  - spot::ltl::postfix\_visitor, 250
- done
  - spot::evtgba\_iterator, 174
  - spot::loopless\_modular\_mixed\_radix\_gray\_code, 208
  - spot::numbered\_state\_heap\_const\_iterator, 223
  - spot::tgba\_explicit\_succ\_iterator, 327
  - spot::tgba\_succ\_iterator, 375
  - spot::tgba\_succ\_iterator\_concrete, 379
  - spot::tgba\_succ\_iterator\_product, 383
- done\_
  - spot::loopless\_modular\_mixed\_radix\_gray\_code, 208
- dotty
  - ltl\_io, 20
- dotty\_decorator
  - spot::dotty\_decorator, 147
- dotty\_reachable
  - evtgbalgorithms/dotty.hh, 419
  - spot, 69
  - tgba\_io, 33
- drand
  - random, 30
- dump
  - ltl\_io, 20
  - spot::bdd\_dict, 96
  - spot::ltl::atomic\_prop, 82
  - spot::ltl::binop, 110
  - spot::ltl::constant, 123
  - spot::ltl::formula, 194
  - spot::ltl::multop, 217
  - spot::ltl::ref\_formula, 258
  - spot::ltl::unop, 407
- dump\_
  - spot::ltl::atomic\_prop, 84
  - spot::ltl::binop, 112
  - spot::ltl::constant, 124
  - spot::ltl::formula, 194
  - spot::ltl::multop, 219
  - spot::ltl::ref\_formula, 259
  - spot::ltl::unop, 408
- dump\_free\_list
  - spot::bdd\_allocator, 89
  - spot::bdd\_dict::annon\_free\_list, 101
  - spot::free\_list, 198
- dump\_instances
  - spot::ltl::atomic\_prop, 82
  - spot::symbol, 292
- dump\_priorities
  - spot::ltl::random\_ltl, 253
- duplicator\_node
  - spot::duplicator\_node, 151
- duplicator\_node\_delayed
  - spot::duplicator\_node\_delayed, 155
- duplicator\_vertice\_
  - spot::parity\_game\_graph, 234
  - spot::parity\_game\_graph\_delayed, 240
  - spot::parity\_game\_graph\_direct, 246
- ec\_statistics
  - spot::ec\_statistics, 159
- ecs\_
  - spot::couvreur99\_check, 128
  - spot::couvreur99\_check\_result, 132
  - spot::couvreur99\_check\_shy, 137
- Emptiness-check algorithms, 42
- Emptiness-check algorithms for SSP, 17
- Emptiness-check statistics, 50
- Emptiness-checks, 41
- emptiness\_check
  - spot::emptiness\_check, 162
- emptiness\_check\_algorithms
  - bit\_state\_hashing\_magic\_search, 42
  - bit\_state\_hashing\_se05\_search, 43
  - explicit\_gv04\_check, 43
  - explicit\_magic\_search, 44
  - explicit\_se05\_search, 45
  - explicit\_tau03\_opt\_search, 46
  - explicit\_tau03\_search, 47
- emptiness\_check\_result
  - spot::emptiness\_check\_result, 164
- emptiness\_check\_ssp
  - couvreur99\_check\_ssp\_semi, 17
  - couvreur99\_check\_ssp\_shy, 17
  - couvreur99\_check\_ssp\_shy\_semi, 17
- empty
  - spot::scc\_stack, 262
  - spot::timer\_map, 394
- end
  - spot::evtgba\_reachable\_iterator, 180
  - spot::evtgba\_reachable\_iterator\_breadth\_first, 183
  - spot::evtgba\_reachable\_iterator\_depth\_first, 187
  - spot::parity\_game\_graph, 233
  - spot::parity\_game\_graph\_delayed, 239
  - spot::parity\_game\_graph\_direct, 245
  - spot::tgba\_reachable\_iterator, 337

- spot::tgba\_reachable\_iterator\_breadth\_first, 341
- spot::tgba\_reachable\_iterator\_depth\_first, 345
- spot::tgba\_reduc, 355
- yy::Location, 206
- yy::Stack, 276
- env
  - spot::ltl::atomic\_prop, 82
- env\_
  - spot::gspn\_interface, 202
  - spot::gspn\_ssp\_interface, 204
  - spot::ltl::atomic\_prop, 84
- Equiv
  - spot::ltl::binop, 110
- err\_
  - spot::gspn\_exeption, 200
- escape\_str
  - misc\_tools, 28
- Essential LTL types, 18
- Essential TGBA types, 31
- evtgba
  - spot::evtgba, 167
- evtgba/ Directory Reference, 50
- evtgba/evtgba.hh, 414
- evtgba/evtgbaiter.hh, 415
- evtgba/explicit.hh, 416
- evtgba/product.hh, 417
- evtgba/symbol.hh, 418
- evtgba\_explicit
  - spot::evtgba\_explicit, 170
- evtgba\_parse
  - evtgbaparse/public.hh, 425
  - spot, 69
- evtgba\_parse\_error
  - evtgbaparse/public.hh, 425
  - spot, 66
- evtgba\_parse\_error\_list
  - evtgbaparse/public.hh, 425
  - spot, 66
- evtgba\_product
  - spot::evtgba\_product, 176
- evtgba\_product\_operands
  - spot::evtgba\_product, 176
- evtgba\_reachable\_iterator
  - spot::evtgba\_reachable\_iterator, 180
- evtgba\_reachable\_iterator\_breadth\_first
  - spot::evtgba\_reachable\_iterator\_breadth\_first, 183
- evtgba\_reachable\_iterator\_depth\_first
  - spot::evtgba\_reachable\_iterator\_depth\_first, 187
- evtgba\_save\_reachable
  - evtgbalgorithms/save.hh, 422
  - spot, 70
- evtgbalgorithms/ Directory Reference, 51
- evtgbalgorithms/dotty.hh, 418
  - dotty\_reachable, 419
- evtgbalgorithms/reachiter.hh, 420
- evtgbalgorithms/save.hh, 422
  - evtgba\_save\_reachable, 422
- evtgbalgorithms/tgba2evtgba.hh, 423
- evtgbaparse/ Directory Reference, 51
- evtgbaparse/public.hh, 424
  - evtgba\_parse, 425
  - evtgba\_parse\_error, 425
  - evtgba\_parse\_error\_list, 425
  - format\_evtgba\_parse\_errors, 425
- explicit\_gv04\_check
  - emptiness\_check\_algorithms, 43
- explicit\_magic\_search
  - emptiness\_check\_algorithms, 44
- explicit\_se05\_search
  - emptiness\_check\_algorithms, 45
- explicit\_tau03\_opt\_search
  - emptiness\_check\_algorithms, 46
- explicit\_tau03\_search
  - emptiness\_check\_algorithms, 47
- extend
  - spot::bdd\_allocator, 89
  - spot::bdd\_dict::annon\_free\_list, 101
  - spot::free\_list, 198
- extvarnum
  - spot::bdd\_allocator, 89
- F
  - spot::ltl::unop, 406
- f0\_max
  - spot::minato\_isop::local\_vars, 211
- f0\_min
  - spot::minato\_isop::local\_vars, 211
- f1\_max
  - spot::minato\_isop::local\_vars, 212
- f1\_min
  - spot::minato\_isop::local\_vars, 212
- f\_
  - spot::loopless\_modular\_mixed\_radix\_gray\_code, 208
- f\_max
  - spot::minato\_isop::local\_vars, 212
- f\_min
  - spot::minato\_isop::local\_vars, 212
- False
  - spot::ltl::constant, 122
- false\_instance
  - spot::ltl::constant, 123
- filename
  - yy::Position, 248
- filter

- spot::bfs\_steps, 104
- finalize
  - spot::bfs\_steps, 104
- find
  - spot::numbered\_state\_heap, 221
  - spot::numbered\_state\_heap\_hash\_map, 227
- find\_state
  - spot::couvreur99\_check\_shy, 136
- finish
  - spot::tgba\_bdd\_concrete\_factory, 308
- first
  - spot::evtgba\_iterator, 174
  - spot::loopless\_modular\_mixed\_radix\_gray\_code, 208
  - spot::ltl::binop, 110
  - spot::numbered\_state\_heap\_const\_iterator, 223
  - spot::tgba\_explicit\_succ\_iterator, 327
  - spot::tgba\_succ\_iterator, 375
  - spot::tgba\_succ\_iterator\_concrete, 379
  - spot::tgba\_succ\_iterator\_product, 383
- first\_
  - spot::ltl::binop, 112
- FirstStep
  - spot::minato\_isop::local\_vars, 211
- fl
  - spot::bdd\_allocator, 90
  - spot::bdd\_dict::annon\_free\_list, 102
  - spot::free\_list, 199
- format\_acceptance\_condition
  - spot::evtgba, 167
  - spot::evtgba\_explicit, 171
  - spot::evtgba\_product, 177
- format\_acceptance\_conditions
  - spot::evtgba, 167
  - spot::evtgba\_explicit, 171
  - spot::evtgba\_product, 177
- format\_evtgba\_parse\_errors
  - evtgbaparse/public.hh, 425
  - spot, 70
- format\_label
  - spot::evtgba, 167
  - spot::evtgba\_explicit, 171
  - spot::evtgba\_product, 177
- format\_parse\_errors
  - ltl\_io, 21
- format\_state
  - spot::evtgba, 167
  - spot::evtgba\_explicit, 171
  - spot::evtgba\_product, 177
  - spot::tgba, 296
  - spot::tgba\_bdd\_concrete, 302
  - spot::tgba\_explicit, 321
  - spot::tgba\_product, 332
  - spot::tgba\_reduc, 355
  - spot::tgba\_sba\_proxy, 370
  - spot::tgba\_tba\_proxy, 388
- format\_tgba\_parse\_errors
  - tgba\_io, 33
- formula2bdd.hh
  - bdd\_to\_formula, 461
  - formula\_to\_bdd, 461
- formula\_to\_bdd
  - formula2bdd.hh, 461
  - spot, 70
- FourthStep
  - spot::minato\_isop::local\_vars, 211
- free\_anonymous\_list\_of
  - spot::bdd\_dict, 98
- free\_anonymous\_list\_of\_type
  - spot::bdd\_dict, 95
- free\_list\_type
  - spot::bdd\_allocator, 88
  - spot::bdd\_dict::annon\_free\_list, 101
  - spot::free\_list, 198
- free\_relation\_simulation
  - tgba\_reduction, 38
- fv\_map
  - spot::bdd\_dict, 95
- G
  - spot::ltl::unop, 406
- g0
  - spot::minato\_isop::local\_vars, 212
- g1
  - spot::minato\_isop::local\_vars, 212
- generate
  - spot::ltl::random\_ltl, 253
- get\_acc
  - spot::duplicator\_node, 151
  - spot::duplicator\_node\_delayed, 156
- get\_acceptance\_condition
  - spot::tgba\_explicit, 321
  - spot::tgba\_reduc, 356
- get\_acceptance\_condition\_visited
  - spot::spoiler\_node\_delayed, 273
- get\_core\_data
  - spot::tgba\_bdd\_concrete, 303
  - spot::tgba\_bdd\_concrete\_factory, 308
  - spot::tgba\_bdd\_factory, 315
- get\_delayed\_relation\_simulation
  - tgba\_reduction, 39
- get\_dict
  - spot::tgba, 296
  - spot::tgba\_bdd\_concrete, 303
  - spot::tgba\_bdd\_concrete\_factory, 308
  - spot::tgba\_explicit, 321
  - spot::tgba\_product, 332

- spot::tgba\_reduc, 356
- spot::tgba\_sba\_proxy, 370
- spot::tgba\_tba\_proxy, 389
- get\_direct\_relation\_simulation
  - tgba\_reduction, 39
- get\_duplicator\_node
  - spot::duplicator\_node, 151
  - spot::duplicator\_node\_delayed, 156
  - spot::spoiler\_node, 269
  - spot::spoiler\_node\_delayed, 273
- get\_err
  - spot::gspn\_exeption, 200
- get\_index
  - spot::numbered\_state\_heap\_const\_iterator, 223
- get\_init\_bdd
  - spot::tgba\_bdd\_concrete, 303
- get\_init\_state
  - spot::tgba, 296
  - spot::tgba\_bdd\_concrete, 303
  - spot::tgba\_explicit, 322
  - spot::tgba\_product, 333
  - spot::tgba\_reduc, 356
  - spot::tgba\_sba\_proxy, 370
  - spot::tgba\_tba\_proxy, 389
- get\_label
  - spot::duplicator\_node, 151
  - spot::duplicator\_node\_delayed, 156
- get\_lead\_2\_acc\_all
  - spot::duplicator\_node\_delayed, 156
  - spot::spoiler\_node\_delayed, 273
- get\_nb\_succ
  - spot::duplicator\_node, 151
  - spot::duplicator\_node\_delayed, 156
  - spot::spoiler\_node, 270
  - spot::spoiler\_node\_delayed, 273
- get\_pair
  - spot::duplicator\_node, 152
  - spot::duplicator\_node\_delayed, 156
  - spot::spoiler\_node, 270
  - spot::spoiler\_node\_delayed, 273
- get\_progress\_measure
  - spot::duplicator\_node\_delayed, 156
  - spot::spoiler\_node\_delayed, 273
- get\_prop\_map
  - spot::ltl::declarative\_environment, 143
- get\_relation
  - spot::parity\_game\_graph, 233
  - spot::parity\_game\_graph\_delayed, 239
  - spot::parity\_game\_graph\_direct, 245
- get\_spoiler\_node
  - spot::duplicator\_node, 152
  - spot::duplicator\_node\_delayed, 156
  - spot::spoiler\_node, 270
- spot::spoiler\_node\_delayed, 273
- get\_state
  - spot::numbered\_state\_heap\_const\_iterator, 223
  - spot::state\_evtgba\_explicit, 282
  - spot::state\_explicit, 285
- get\_where
  - spot::gspn\_exeption, 200
- group\_
  - spot::couvereur99\_check\_shy, 137
- gspn/ Directory Reference, 51
- gspn/common.hh, 412
- gspn/gspn.hh, 413
- gspn/ssp.hh, 414
- gspn\_exeption
  - spot::gspn\_exeption, 200
- gspn\_interface
  - spot::gspn\_interface, 201
- gspn\_ssp\_interface
  - spot::gspn\_ssp\_interface, 203
- h
  - spot::couvereur99\_check\_status, 141
  - spot::numbered\_state\_heap\_hash\_map, 228
- h\_
  - spot::tgba\_reduc, 361
- has\_acceptance\_condition
  - spot::tgba\_explicit, 322
  - spot::tgba\_reduc, 356
- has\_state
  - spot::connected\_component\_hash\_set, 117
  - spot::explicit\_connected\_component, 190
- hash
  - spot::ltl::atomic\_prop, 82
  - spot::ltl::binop, 110
  - spot::ltl::constant, 123
  - spot::ltl::formula, 194
  - spot::ltl::multop, 217
  - spot::ltl::ref\_formula, 258
  - spot::ltl::unop, 407
  - spot::state, 278
  - spot::state\_bdd, 280
  - spot::state\_evtgba\_explicit, 282
  - spot::state\_explicit, 285
  - spot::state\_product, 288
- hash\_funcs
  - wang32\_hash, 29
- hash\_key\_
  - spot::ltl::atomic\_prop, 84
  - spot::ltl::binop, 112
  - spot::ltl::constant, 124
  - spot::ltl::formula, 194
  - spot::ltl::multop, 219
  - spot::ltl::ref\_formula, 259



- spot::ltl::unop, 408
- hash\_type
  - spot::numbered\_state\_heap\_hash\_map, 226
- Hashing functions, 28
- height
  - yy::Stack, 276
- i\_
  - spot::tgba\_explicit\_succ\_iterator, 328
- Implies
  - spot::ltl::binop, 109
- implies
  - spot::duplicator\_node, 152
  - spot::duplicator\_node\_delayed, 156
- implies\_acc
  - spot::duplicator\_node\_delayed, 156
- implies\_label
  - spot::duplicator\_node\_delayed, 156
- in
  - spot::evtgba\_explicit::state, 173
  - spot::evtgba\_explicit::transition, 173
- inc\_ars\_cycle\_states
  - spot::ars\_statistics, 77
  - spot::couvreur99\_check\_result, 132
- inc\_ars\_prefix\_states
  - spot::ars\_statistics, 78
  - spot::couvreur99\_check\_result, 132
- inc\_depth
  - spot::couvreur99\_check, 128
  - spot::couvreur99\_check\_shy, 136
  - spot::ec\_statistics, 159
- inc\_states
  - spot::couvreur99\_check, 128
  - spot::couvreur99\_check\_shy, 136
  - spot::ec\_statistics, 159
- inc\_transitions
  - spot::couvreur99\_check, 128
  - spot::couvreur99\_check\_shy, 136
  - spot::ec\_statistics, 159
- inc\_weight\_handler
  - spot::weight, 411
- index
  - spot::connected\_component\_hash\_set, 117
  - spot::explicit\_connected\_component, 190
  - spot::numbered\_state\_heap, 222
  - spot::numbered\_state\_heap\_hash\_map, 227
  - spot::scc\_stack::connected\_component, 264
- index\_and\_insert
  - spot::numbered\_state\_heap, 222
  - spot::numbered\_state\_heap\_hash\_map, 227
- init\_
  - spot::tgba\_bdd\_concrete, 305
  - spot::tgba\_explicit, 324
  - spot::tgba\_reduc, 361
- init\_iter
  - spot::evtgba, 167
  - spot::evtgba\_explicit, 171
  - spot::evtgba\_product, 177
- init\_states\_
  - spot::evtgba\_explicit, 172
- initial\_column
  - yy::Position, 248
- initial\_line
  - yy::Position, 248
- initialize
  - spot::bdd\_allocator, 89
  - spot::bdd\_dict, 96
- initialized
  - spot::bdd\_allocator, 90
  - spot::bdd\_dict, 98
- Input/Output of LTL formulae, 19
- Input/Output of TGBA, 32
- insert
  - spot::bdd\_allocator, 89
  - spot::bdd\_dict::annon\_free\_list, 101
  - spot::connected\_component\_hash\_set, 117
  - spot::explicit\_connected\_component, 190
  - spot::free\_list, 199
  - spot::numbered\_state\_heap, 222
  - spot::numbered\_state\_heap\_hash\_map, 227
- instance
  - spot::connected\_component\_hash\_set - factory, 119
  - spot::dotty\_decorator, 147
  - spot::ltl::atomic\_prop, 83
  - spot::ltl::binop, 110
  - spot::ltl::default\_environment, 145
  - spot::ltl::multop, 217
  - spot::ltl::unop, 407
  - spot::numbered\_state\_heap\_hash\_map - factory, 229
  - spot::symbol, 292
  - spot::tgba\_run\_dotty\_decorator, 366
- instance\_count
  - spot::ltl::atomic\_prop, 83
  - spot::ltl::binop, 110
  - spot::ltl::multop, 218
  - spot::ltl::unop, 407
  - spot::symbol, 292
- instances
  - spot::ltl::atomic\_prop, 84
  - spot::ltl::binop, 112
  - spot::ltl::multop, 219
  - spot::ltl::unop, 408
- instances\_
  - spot::symbol, 292
- is\_bare\_word
  - misc\_tools, 28



- is\_eventual
  - ltl\_misc, 25
- is\_FG
  - ltl\_misc, 25
- is\_GF
  - ltl\_misc, 26
- is\_not\_accepting
  - spot::tgba\_reduc, 356
- is\_registered\_acceptance\_variable
  - spot::bdd\_dict, 96
- is\_registered\_proposition
  - spot::bdd\_dict, 96
- is\_registered\_state
  - spot::bdd\_dict, 96
- is\_terminal
  - spot::tgba\_reduc, 356
- is\_universal
  - ltl\_misc, 26
- item\_type
  - spot::timer\_map, 394
- Iterator
  - yy::Stack, 275
- iterator
  - spot::numbered\_state\_heap, 222
  - spot::numbered\_state\_heap\_hash\_map, 228
- label
  - spot::evtgba\_explicit::transition, 174
  - spot::tgba\_run::step, 363
- label\_
  - spot::duplicator\_node, 152
  - spot::duplicator\_node\_delayed, 157
- last
  - spot::loopless\_modular\_mixed\_radix\_gray\_-code, 208
- last\_support\_conditions\_input\_
  - spot::tgba, 298
- last\_support\_conditions\_output\_
  - spot::tgba, 298
- last\_support\_variables\_input\_
  - spot::tgba, 298
- last\_support\_variables\_output\_
  - spot::tgba, 298
- lbtt\_reachable
  - tgba\_io, 34
- lead\_2\_acc\_all\_
  - spot::duplicator\_node\_delayed, 157
  - spot::spoiler\_node\_delayed, 274
- left
  - spot::state\_product, 288
- left\_
  - spot::state\_product, 288
  - spot::tgba\_product, 335
  - spot::tgba\_succ\_iterator\_product, 384
- left\_acc\_complement\_
  - spot::tgba\_product, 335
- left\_neg\_
  - spot::tgba\_succ\_iterator\_product, 384
- length
  - ltl\_misc, 26
- lift
  - spot::parity\_game\_graph, 233
  - spot::parity\_game\_graph\_delayed, 239
  - spot::parity\_game\_graph\_direct, 245
- line
  - yy::Position, 248
- lines
  - yy::Location, 205
  - yy::Position, 248
- link\_decl
  - spot::dotty\_decorator, 147
  - spot::tgba\_run\_dotty\_decorator, 366
- lnode\_pred
  - spot::duplicator\_node, 152
  - spot::duplicator\_node\_delayed, 157
  - spot::spoiler\_node, 270
  - spot::spoiler\_node\_delayed, 274
- lnode\_succ
  - spot::duplicator\_node, 152
  - spot::duplicator\_node\_delayed, 157
  - spot::spoiler\_node, 270
  - spot::spoiler\_node\_delayed, 274
- local\_vars
  - spot::minato\_isop::local\_vars, 211
- Location
  - yy::Location, 205
- location.hh
  - operator+, 438
  - operator+=", 438
  - operator<<, 439
- loopless\_modular\_mixed\_radix\_gray\_code
  - spot::loopless\_modular\_mixed\_radix\_gray\_-code, 207
- LTL Abstract Syntax Tree, 18
- LTL environments, 19
- LTL formulae, 17
- ltl\_essential
  - clone, 18
  - destroy, 18
- ltl\_io
  - dotty, 20
  - dump, 20
  - format\_parse\_errors, 21
  - parse, 21
  - parse\_error, 20
  - parse\_error\_list, 20
  - to\_spin\_string, 21
  - to\_string, 22

- ltl\_misc
  - atomic\_prop\_collect, 25
  - atomic\_prop\_set, 25
  - is\_eventual, 25
  - is\_FG, 25
  - is\_GF, 26
  - is\_universal, 26
  - length, 26
  - syntactic\_implication, 26
  - syntactic\_implication\_neg, 26
- ltl\_rewriting
  - Reduce\_All, 23
  - Reduce\_Basics, 23
  - Reduce\_Eventuality\_And\_Universality, 23
  - Reduce\_None, 23
  - Reduce\_Syntactic\_Implications, 23
- ltl\_rewriting
  - basic\_reduce, 23
  - negative\_normal\_form, 23
  - reduce, 24
  - reduce\_options, 23
  - simplify\_f\_g, 24
  - unabbreviate\_logic, 24
- ltl\_to\_tgba\_fm
  - tgba\_ltl, 35
- ltl\_to\_tgba\_lacim
  - tgba\_ltl, 36
- ltlast/ Directory Reference, 52
- ltlast/allnodes.hh, 429
- ltlast/atomic\_prop.hh, 429
- ltlast/binop.hh, 430
- ltlast/constant.hh, 431
- ltlast/formula.hh, 431
- ltlast/multop.hh, 433
- ltlast/predecl.hh, 433
- ltlast/refformula.hh, 434
- ltlast/unop.hh, 434
- ltlast/visitor.hh, 435
- ltlenv/ Directory Reference, 53
- ltlenv/declenv.hh, 436
- ltlenv/defaultenv.hh, 436
- ltlenv/environment.hh, 437
- ltlparse/ Directory Reference, 54
- ltlparse/location.hh, 437
- ltlparse/position.hh, 439
- ltlparse/public.hh, 426
- ltlparse/stack.hh, 440
- ltlvisit/ Directory Reference, 54
- ltlvisit/apcollect.hh, 441
- ltlvisit/basicreduce.hh, 441
- ltlvisit/clone.hh, 442
- ltlvisit/destroy.hh, 443
- ltlvisit/dotty.hh, 419
- ltlvisit/dump.hh, 443
- ltlvisit/length.hh, 444
- ltlvisit/lunabbrev.hh, 444
- ltlvisit/nenoform.hh, 445
- ltlvisit/postfix.hh, 445
- ltlvisit/randomltl.hh, 446
- ltlvisit/reduce.hh, 446
- ltlvisit/simpfg.hh, 447
- ltlvisit/syntimpl.hh, 447
- ltlvisit/tostring.hh, 448
- ltlvisit/tunabbrev.hh, 449
- lvarnum
  - spot::bdd\_allocator, 90
  - spot::bdd\_dict, 98
- m
  - spot::weight, 412
- m\_
  - spot::barand, 85
  - spot::loopless\_modular\_mixed\_radix\_gray\_code, 208
- mainpage.dox, 412
- map
  - spot::ltl::atomic\_prop, 82
  - spot::ltl::binop, 109
  - spot::ltl::multop, 216
  - spot::ltl::unop, 406
  - spot::symbol, 292
- map\_
  - spot::tgba\_run\_dotty\_decorator, 366
- match
  - spot::bfs\_steps, 105
  - spot::duplicator\_node, 152
  - spot::duplicator\_node\_delayed, 156
- max\_depth
  - spot::couvereur99\_check, 128
  - spot::couvereur99\_check\_shy, 136
  - spot::ec\_statistics, 159
- max\_depth\_
  - spot::ec\_statistics, 159
- merge\_state
  - spot::tgba\_reduc, 356
- merge\_state\_delayed
  - spot::tgba\_reduc, 357
- merge\_transitions
  - spot::tgba\_explicit, 322
  - spot::tgba\_reduc, 357
- min\_n
  - spot::ltl::random\_ltl::op\_proba, 255
- minato\_isop
  - spot::minato\_isop, 210
- misc/ Directory Reference, 55
- misc/bareword.hh, 449
- misc/bddalloc.hh, 450
- misc/bddlt.hh, 450

- misc/escape.hh, 451
- misc/freelist.hh, 451
- misc/hash.hh, 452
- misc/hashfunc.hh, 452
- misc/minato.hh, 453
- misc/modgray.hh, 453
- misc/random.hh, 453
- misc/timer.hh, 454
- misc/version.hh, 455
- misc\_tools
  - escape\_str, 28
  - is\_bare\_word, 28
  - quote\_unless\_bare\_word, 28
  - version, 28
- Miscellaneous algorithms for LTL formulae, 24
- Miscellaneous algorithms on TGBA, 39
- Miscellaneous helper algorithms, 27
- mrnd
  - random, 30
- multop
  - spot::ltl::multop, 217
- n
  - spot::couvreur99\_check\_shy::todo\_item, 139
- n\_
  - spot::barand, 85
  - spot::loopless\_modular\_mixed\_radix\_gray\_code, 208
- name
  - spot::ltl::atomic\_prop, 83
  - spot::ltl::declarative\_environment, 143
  - spot::ltl::default\_environment, 145
  - spot::ltl::environment, 165
  - spot::ltl::random\_ltl::op\_proba, 255
  - spot::symbol, 292
- name\_
  - spot::ltl::atomic\_prop, 84
  - spot::symbol, 292
- name\_state\_map\_
  - spot::evtgba\_explicit, 172
  - spot::tgba\_explicit, 324
  - spot::tgba\_reduc, 361
- nb\_node\_parity\_game
  - spot::parity\_game\_graph, 234
  - spot::parity\_game\_graph\_delayed, 240
  - spot::parity\_game\_graph\_direct, 246
- nb\_set\_acc\_cond
  - spot::parity\_game\_graph\_delayed, 239
- neg\_acceptance\_conditions
  - spot::tgba, 296
  - spot::tgba\_bdd\_concrete, 303
  - spot::tgba\_explicit, 322
  - spot::tgba\_product, 333
  - spot::tgba\_reduc, 357
  - spot::tgba\_sba\_proxy, 371
  - spot::tgba\_tba\_proxy, 389
- neg\_acceptance\_conditions\_
  - spot::tgba\_explicit, 324
  - spot::tgba\_product, 335
  - spot::tgba\_reduc, 361
- neg\_all\_acc
  - spot::weight, 412
- negacc\_set
  - spot::tgba\_bdd\_core\_data, 313
- negative\_normal\_form
  - ltl\_rewriting, 23
- never\_claim\_reachable
  - tgba\_io, 34
- next
  - spot::evtgba\_iterator, 174
  - spot::loopless\_modular\_mixed\_radix\_gray\_code, 208
  - spot::minato\_isop, 210
  - spot::numbered\_state\_heap\_const\_iterator, 223
  - spot::tgba\_explicit\_succ\_iterator, 328
  - spot::tgba\_succ\_iterator, 376
  - spot::tgba\_succ\_iterator\_concrete, 379
  - spot::tgba\_succ\_iterator\_product, 384
- next\_non\_false\_
  - spot::tgba\_succ\_iterator\_product, 384
- next\_set
  - spot::tgba\_bdd\_core\_data, 313
- next\_state
  - spot::evtgba\_reachable\_iterator, 180
  - spot::evtgba\_reachable\_iterator\_breadth\_first, 184
  - spot::evtgba\_reachable\_iterator\_depth\_first, 187
  - spot::parity\_game\_graph, 233
  - spot::parity\_game\_graph\_delayed, 239
  - spot::parity\_game\_graph\_direct, 245
  - spot::tgba\_reachable\_iterator, 337
  - spot::tgba\_reachable\_iterator\_breadth\_first, 342
  - spot::tgba\_reachable\_iterator\_depth\_first, 345
  - spot::tgba\_reduc, 357
- next\_to\_now
  - spot::bdd\_dict, 98
- non\_one\_radixes\_
  - spot::loopless\_modular\_mixed\_radix\_gray\_code, 209
- Not
  - spot::ltl::unop, 406
- not\_win
  - spot::duplicator\_node, 152
  - spot::duplicator\_node\_delayed, 157
  - spot::spoiler\_node, 270

- spot::spoiler\_node\_delayed, 274
- notacc\_set
  - spot::tgba\_bdd\_core\_data, 313
- notnext\_set
  - spot::tgba\_bdd\_core\_data, 313
- notnow\_set
  - spot::tgba\_bdd\_core\_data, 313
- notvar\_set
  - spot::tgba\_bdd\_core\_data, 314
- now\_formula\_map
  - spot::bdd\_dict, 98
- now\_map
  - spot::bdd\_dict, 98
- now\_set
  - spot::tgba\_bdd\_core\_data, 314
- now\_to\_next
  - spot::bdd\_dict, 98
- nownext\_set
  - spot::tgba\_bdd\_core\_data, 314
- nrand
  - random, 30
- ns\_map
  - spot::evtgba\_explicit, 170
  - spot::tgba\_explicit, 320
  - spot::tgba\_reduc, 353
- nth
  - spot::ltl::multop, 218
- num
  - spot::couvreur99\_check\_shy, 137
- num\_
  - spot::duplicator\_node, 152
  - spot::duplicator\_node\_delayed, 157
  - spot::spoiler\_node, 270
  - spot::spoiler\_node\_delayed, 274
- num\_acc\_
  - spot::tgba, 298
- number\_of\_acceptance\_conditions
  - spot::tgba, 297
  - spot::tgba\_bdd\_concrete, 303
  - spot::tgba\_explicit, 322
  - spot::tgba\_product, 333
  - spot::tgba\_reduc, 357
  - spot::tgba\_sba\_proxy, 371
  - spot::tgba\_tba\_proxy, 389
- numbered\_state\_heap\_hash\_map\_factory
  - spot::numbered\_state\_heap\_hash\_map\_-factory, 229
- op
  - spot::ltl::binop, 110
  - spot::ltl::multop, 218
  - spot::ltl::unop, 407
- op\_
  - spot::evtgba\_product, 178
- spot::ltl::binop, 112
- spot::ltl::multop, 219
- spot::ltl::unop, 408
- op\_name
  - spot::ltl::binop, 111
  - spot::ltl::multop, 218
  - spot::ltl::unop, 407
- operator const symbol \*
  - spot::rsymbol, 261
- operator!=
  - spot::rsymbol, 261
- operator()
  - spot::bdd\_less\_than, 103
  - spot::ltl::formula\_ptr\_hash, 195
  - spot::ltl::formula\_ptr\_less\_than, 196
  - spot::ltl::multop::paircmp, 220
  - spot::ptr\_hash, 251
  - spot::state\_ptr\_equal, 289
  - spot::state\_ptr\_hash, 290
  - spot::state\_ptr\_less\_than, 290
  - spot::string\_hash, 291
- operator+
  - location.hh, 438
  - position.hh, 440
  - yy, 75
- operator+=
  - location.hh, 438
  - position.hh, 440
  - spot::weight, 411
  - yy, 75
- operator-
  - position.hh, 440
  - spot::weight, 411
  - yy, 75
- operator-=
  - position.hh, 440
  - spot::weight, 411
  - yy, 75
- operator<
  - spot::rsymbol, 261
- operator<<
  - common.hh, 413
  - location.hh, 439
  - position.hh, 440
  - spot, 70
  - spot::weight, 412
  - yy, 75
- operator=
  - spot::bdd\_dict, 96
  - spot::rsymbol, 261
  - spot::tgba\_bdd\_core\_data, 312
  - spot::tgba\_run, 363
- operator==
  - spot::rsymbol, 261

- operator[]
  - yy::Slice, 267
  - yy::Stack, 276
- Or
  - spot::ltl::multop, 217
- out
  - spot::evtgba\_explicit::state, 173
  - spot::evtgba\_explicit::transition, 174
- pair
  - spot::ltl::atomic\_prop, 82
  - spot::ltl::binop, 109
  - spot::ltl::multop, 216
  - spot::ltl::unop, 406
- pairf
  - spot::ltl::binop, 109
- parity\_game\_graph
  - spot::parity\_game\_graph, 232
- parity\_game\_graph\_delayed
  - spot::parity\_game\_graph\_delayed, 238
- parity\_game\_graph\_direct
  - spot::parity\_game\_graph\_direct, 245
- parse
  - ltl\_io, 21
- parse\_error
  - ltl\_io, 20
- parse\_error\_list
  - ltl\_io, 20
- parse\_options
  - spot::ltl::random\_ltl, 253
- pm
  - spot::weight, 412
- pop
  - spot::scc\_stack, 262
  - yy::Stack, 276
- poprem\_
  - spot::couvreur99\_check, 128
  - spot::couvreur99\_check\_shy, 137
- pos\_lenght\_pair
  - spot::bdd\_allocator, 88
  - spot::bdd\_dict::annon\_free\_list, 101
  - spot::free\_list, 198
- Position
  - yy::Position, 248
- position.hh
  - operator+, 440
  - operator+=, 440
  - operator-, 440
  - operator=, 440
  - operator<<, 440
- postfix\_visitor
  - spot::ltl::postfix\_visitor, 250
- prand
  - random, 30
- pred\_iter
  - spot::evtgba, 168
  - spot::evtgba\_explicit, 171, 172
  - spot::evtgba\_product, 177
- prefix
  - spot::tgba\_run, 363
- prefix\_states\_
  - spot::ars\_statistics, 78
- print
  - spot::parity\_game\_graph, 233
  - spot::parity\_game\_graph\_delayed, 239
  - spot::parity\_game\_graph\_direct, 245
  - spot::timer\_map, 394
- print\_stats
  - spot::couvreur99\_check, 128
  - spot::couvreur99\_check\_result, 132
  - spot::couvreur99\_check\_shy, 136
  - spot::couvreur99\_check\_status, 141
  - spot::emptiness\_check, 162
- print\_tgba\_run
  - tgba\_run, 48
- proba
  - spot::ltl::random\_ltl::op\_proba, 255
- proba\_
  - spot::ltl::random\_ltl, 254
- proba\_2\_
  - spot::ltl::random\_ltl, 254
- process\_link
  - spot::evtgba\_reachable\_iterator, 180
  - spot::evtgba\_reachable\_iterator\_breadth\_first, 184
  - spot::evtgba\_reachable\_iterator\_depth\_first, 187
  - spot::parity\_game\_graph, 233
  - spot::parity\_game\_graph\_delayed, 239, 240
  - spot::parity\_game\_graph\_direct, 245, 246
  - spot::tgba\_reachable\_iterator, 338
  - spot::tgba\_reachable\_iterator\_breadth\_first, 342
  - spot::tgba\_reachable\_iterator\_depth\_first, 346
  - spot::tgba\_reduc, 357
- process\_state
  - spot::evtgba\_reachable\_iterator, 181
  - spot::evtgba\_reachable\_iterator\_breadth\_first, 184
  - spot::evtgba\_reachable\_iterator\_depth\_first, 188
  - spot::parity\_game\_graph, 234
  - spot::parity\_game\_graph\_delayed, 240
  - spot::parity\_game\_graph\_direct, 246
  - spot::tgba\_reachable\_iterator, 338
  - spot::tgba\_reachable\_iterator\_breadth\_first, 342
  - spot::tgba\_reachable\_iterator\_depth\_first, 346

- spot::tgba\_reduc, 358
- product
  - tgba\_algorithms, 32
- progress\_measure\_
  - spot::duplicator\_node\_delayed, 157
  - spot::spoiler\_node\_delayed, 274
- project\_state
  - spot::tgba, 297
  - spot::tgba\_bdd\_concrete, 304
  - spot::tgba\_explicit, 322
  - spot::tgba\_product, 333
  - spot::tgba\_reduc, 358
  - spot::tgba\_sba\_proxy, 371
  - spot::tgba\_tba\_proxy, 389
- project\_tgba\_run
  - tgba\_run, 49
- prop\_map
  - spot::ltl::declarative\_environment, 143
- props\_
  - spot::ltl::declarative\_environment, 144
- prune
  - spot::duplicator\_node, 152
  - spot::duplicator\_node\_delayed, 156
  - spot::spoiler\_node, 270
  - spot::spoiler\_node\_delayed, 273
- prune\_acc
  - spot::tgba\_reduc, 358
- prune\_scc
  - spot::tgba\_reduc, 358
- push
  - spot::scc\_stack, 262
  - yy::Stack, 276
- q
  - spot::couvereur99\_check\_shy::todo\_item, 139
- quote\_unless\_bare\_word
  - misc\_tools, 28
- quotient\_state
  - spot::tgba\_reduc, 358
- R
  - spot::ltl::binop, 110
- rand
  - spot::barand, 85
- random
  - bmrand, 30
  - drand, 30
  - mrnd, 30
  - nrnd, 30
  - prand, 30
  - rrand, 30
  - srnd, 30
- Random functions, 29
- random\_graph
  - tgba\_misc, 40
- random\_ltl
  - spot::ltl::random\_ltl, 253
- range\_
  - yy::Slice, 267
- recurse
  - spot::ltl::clone\_visitor, 114
  - spot::ltl::simplify\_f\_g\_visitor, 266
  - spot::ltl::unabbreviate\_logic\_visitor, 398
  - spot::ltl::unabbreviate\_ltl\_visitor, 401
- redirect\_transition
  - spot::tgba\_reduc, 358
- reduc\_tgba\_sim
  - tgba\_reduction, 39
- reduce
  - ltl\_rewriting, 24
- Reduce\_All
  - ltl\_rewriting, 23
  - tgba\_reduction, 38
- Reduce\_Basics
  - ltl\_rewriting, 23
- Reduce\_Eventuality\_And\_Universality
  - ltl\_rewriting, 23
- Reduce\_None
  - ltl\_rewriting, 23
  - tgba\_reduction, 38
- reduce\_options
  - ltl\_rewriting, 23
- Reduce\_quotient\_Del\_Sim
  - tgba\_reduction, 38
- Reduce\_quotient\_Dir\_Sim
  - tgba\_reduction, 38
- reduce\_run
  - tgba\_run, 49
- Reduce\_Scc
  - tgba\_reduction, 38
- Reduce\_Syntactic\_Implications
  - ltl\_rewriting, 23
- reduce\_tgba\_options
  - tgba\_reduction, 38
- Reduce\_transition\_Del\_Sim
  - tgba\_reduction, 38
- Reduce\_transition\_Dir\_Sim
  - tgba\_reduction, 38
- ref
  - spot::ltl::atomic\_prop, 83
  - spot::ltl::binop, 111
  - spot::ltl::constant, 123
  - spot::ltl::formula, 194
  - spot::ltl::multop, 218
  - spot::ltl::ref\_formula, 258
  - spot::ltl::unop, 407
  - spot::symbol, 292
- ref\_

- spot::ltl::atomic\_prop, 83
- spot::ltl::binop, 111
- spot::ltl::constant, 123
- spot::ltl::formula, 194
- spot::ltl::multop, 218
- spot::ltl::ref\_formula, 258
- spot::ltl::unop, 407
- ref\_count\_
  - spot::ltl::atomic\_prop, 83
  - spot::ltl::binop, 111
  - spot::ltl::multop, 218
  - spot::ltl::ref\_formula, 258
  - spot::ltl::unop, 408
  - spot::symbol, 292
- ref\_counter\_
  - spot::ltl::ref\_formula, 259
- ref\_formula
  - spot::ltl::ref\_formula, 258
- ref\_set
  - spot::bdd\_dict, 95
- refs\_
  - spot::symbol, 292
- register\_acceptance\_variable
  - spot::bdd\_dict, 96
- register\_acceptance\_variables
  - spot::bdd\_dict, 96
- register\_all\_variables\_of
  - spot::bdd\_dict, 97
- register\_anonymous\_variables
  - spot::bdd\_dict, 97
- register\_n
  - spot::bdd\_allocator, 89
  - spot::bdd\_dict::annon\_free\_list, 101
  - spot::free\_list, 199
- register\_proposition
  - spot::bdd\_dict, 97
- register\_propositions
  - spot::bdd\_dict, 97
- register\_state
  - spot::bdd\_dict, 97
- relation
  - spot::tgba\_bdd\_core\_data, 314
- release\_n
  - spot::bdd\_allocator, 89
  - spot::bdd\_dict::annon\_free\_list, 102
  - spot::free\_list, 199
- release\_variables
  - spot::bdd\_allocator, 90
  - spot::bdd\_dict, 97
- rem
  - spot::connected\_component\_hash\_set, 117
  - spot::explicit\_connected\_component, 190
  - spot::scc\_stack, 262
  - spot::scc\_stack::connected\_component, 264
- remove
  - spot::bdd\_allocator, 90
  - spot::bdd\_dict::annon\_free\_list, 102
  - spot::free\_list, 199
- remove\_acc
  - spot::tgba\_reduc, 359
- remove\_component
  - spot::couvreur99\_check, 128
  - spot::couvreur99\_check\_shy, 137
  - spot::tgba\_reduc, 359
- remove\_predecessor\_state
  - spot::tgba\_reduc, 359
- remove\_scc
  - spot::tgba\_reduc, 359
- remove\_state
  - spot::tgba\_reduc, 359
- replay\_tgba\_run
  - tgba\_run, 49
- require
  - spot::ltl::declarative\_environment, 143
  - spot::ltl::default\_environment, 145
  - spot::ltl::environment, 165
- result
  - spot::couvreur99\_check, 128
  - spot::couvreur99\_check\_shy, 137
  - spot::ltl::clone\_visitor, 114
  - spot::ltl::simplify\_f\_g\_visitor, 266
  - spot::ltl::unabbreviate\_logic\_visitor, 398
  - spot::ltl::unabbreviate\_ltl\_visitor, 401
- result\_
  - spot::ltl::clone\_visitor, 115
  - spot::ltl::simplify\_f\_g\_visitor, 266
  - spot::ltl::unabbreviate\_logic\_visitor, 399
  - spot::ltl::unabbreviate\_ltl\_visitor, 402
- ret\_
  - spot::minato\_isop, 210
- Rewriting LTL formulae, 22
- right
  - spot::state\_product, 288
- right\_
  - spot::state\_product, 288
  - spot::tgba\_product, 335
  - spot::tgba\_succ\_iterator\_product, 384
- right\_acc\_complement\_
  - spot::tgba\_product, 335
- right\_neg\_
  - spot::tgba\_succ\_iterator\_product, 384
- root
  - spot::couvreur99\_check\_status, 141
- root\_
  - spot::tgba\_reduc, 361
- rrand
  - random, 30
- rsymbol

- spot::rsymbol, 260
- rsymbol\_set
  - spot, 66
  - symbol.hh, 418
- run
  - spot::evtgba\_reachable\_iterator, 181
  - spot::evtgba\_reachable\_iterator\_breadth\_first, 184
  - spot::evtgba\_reachable\_iterator\_depth\_first, 188
  - spot::parity\_game\_graph, 234
  - spot::parity\_game\_graph\_delayed, 240
  - spot::parity\_game\_graph\_direct, 246
  - spot::tgba\_reachable\_iterator, 338
  - spot::tgba\_reachable\_iterator\_breadth\_first, 342
  - spot::tgba\_reachable\_iterator\_depth\_first, 346
  - spot::tgba\_reduc, 359
- run\_
  - spot::couvreur99\_check\_result, 132
  - spot::tgba\_run\_dotty\_decorator, 366
- s
  - spot::couvreur99\_check\_shy::successor, 138
  - spot::couvreur99\_check\_shy::todo\_item, 139
  - spot::scc\_stack, 263
  - spot::tgba\_run::step, 364
- s\_
  - spot::barand, 85
  - spot::loopless\_modular\_mixed\_radix\_gray\_code, 209
  - spot::rsymbol, 261
  - spot::tgba\_explicit\_succ\_iterator, 328
- s\_v
  - tgba\_reduction, 38
- sc\_
  - spot::duplicator\_node, 152
  - spot::duplicator\_node\_delayed, 157
  - spot::spoiler\_node, 270
  - spot::spoiler\_node\_delayed, 274
- scc\_computed\_
  - spot::tgba\_reduc, 361
- search
  - spot::bfs\_steps, 105
- second
  - spot::ltl::binop, 111
- second\_
  - spot::ltl::binop, 112
- SecondStep
  - spot::minato\_isop::local\_vars, 211
- seen
  - spot::evtgba\_reachable\_iterator, 181
  - spot::evtgba\_reachable\_iterator\_breadth\_first, 184
  - spot::evtgba\_reachable\_iterator\_depth\_first, 188
  - spot::parity\_game\_graph, 234
  - spot::parity\_game\_graph\_delayed, 241
  - spot::parity\_game\_graph\_direct, 246
  - spot::tgba\_reachable\_iterator, 338
  - spot::tgba\_reachable\_iterator\_breadth\_first, 343
  - spot::tgba\_reachable\_iterator\_depth\_first, 346
  - spot::tgba\_reduc, 361
- seen\_
  - spot::duplicator\_node\_delayed, 157
  - spot::spoiler\_node\_delayed, 274
  - spot::tgba\_reduc, 361
- seen\_map
  - spot::evtgba\_reachable\_iterator, 180
  - spot::evtgba\_reachable\_iterator\_breadth\_first, 183
  - spot::evtgba\_reachable\_iterator\_depth\_first, 187
  - spot::parity\_game\_graph, 232
  - spot::parity\_game\_graph\_delayed, 238
  - spot::parity\_game\_graph\_direct, 244
  - spot::tgba\_reachable\_iterator, 337
  - spot::tgba\_reachable\_iterator\_breadth\_first, 341
  - spot::tgba\_reachable\_iterator\_depth\_first, 345
  - spot::tgba\_reduc, 353
- seq\_
  - yy::Stack, 276
- set\_init\_state
  - spot::evtgba\_explicit, 172
  - spot::tgba\_bdd\_concrete, 304
  - spot::tgba\_explicit, 322
  - spot::tgba\_reduc, 359
- set\_key\_
  - spot::ltl::atomic\_prop, 83
  - spot::ltl::binop, 111
  - spot::ltl::constant, 123
  - spot::ltl::formula, 194
  - spot::ltl::multop, 218
  - spot::ltl::ref\_formula, 259
  - spot::ltl::unop, 408
- set\_lead\_2\_acc\_all
  - spot::duplicator\_node\_delayed, 156
  - spot::spoiler\_node\_delayed, 273
- set\_states
  - spot::couvreur99\_check, 128
  - spot::couvreur99\_check\_shy, 137
  - spot::ec\_statistics, 159
- set\_type
  - spot::connected\_component\_hash\_set, 117
- set\_win
  - spot::duplicator\_node, 152



- spot::duplicator\_node\_delayed, 156
- spot::spoiler\_node, 270
- spot::spoiler\_node\_delayed, 273
- setup
  - spot::ltl::random\_ltl::op\_proba, 255
- si\_
  - spot::tgba\_reduc, 361
- simplify\_f\_g
  - ltl\_rewriting, 24
- simplify\_f\_g\_visitor
  - spot::ltl::simplify\_f\_g\_visitor, 266
- simulation\_relation
  - spot, 66
  - tgbareduc.hh, 472
- size
  - spot::ltl::multop, 219
  - spot::numbered\_state\_heap, 222
  - spot::numbered\_state\_heap\_hash\_map, 228
  - spot::scc\_stack, 262
- Slice
  - yy::Slice, 267
- sn\_map
  - spot::evtgba\_explicit, 170
  - spot::tgba\_explicit, 320
  - spot::tgba\_reduc, 353
- sn\_v
  - tgba\_reduction, 38
- sp\_map
  - spot::tgba\_reduc, 353
- spoiler\_node
  - spot::spoiler\_node, 269
- spoiler\_node\_delayed
  - spot::spoiler\_node\_delayed, 272
- spoiler\_vertice\_
  - spot::parity\_game\_graph, 234
  - spot::parity\_game\_graph\_delayed, 241
  - spot::parity\_game\_graph\_direct, 247
- spot, 57
  - bdd\_format\_accset, 67
  - bdd\_format\_formula, 67
  - bdd\_format\_sat, 67
  - bdd\_format\_set, 67
  - bdd\_print\_acc, 67
  - bdd\_print\_accset, 68
  - bdd\_print\_dot, 68
  - bdd\_print\_formula, 68
  - bdd\_print\_sat, 68
  - bdd\_print\_set, 69
  - bdd\_print\_table, 69
  - bdd\_to\_formula, 69
  - dotty\_reachable, 69
  - evtgba\_parse, 69
  - evtgba\_parse\_error, 66
  - evtgba\_parse\_error\_list, 66
  - evtgba\_save\_reachable, 70
  - format\_evtgba\_parse\_errors, 70
  - formula\_to\_bdd, 70
  - operator<<, 70
  - rsymbol\_set, 66
  - simulation\_relation, 66
  - state\_couple, 66
  - symbol\_set, 66
  - tgba\_to\_evtgba, 70
- spot::acss\_statistics, 76
  - acss\_states, 76
- spot::ars\_statistics, 76
  - ars\_cycle\_states, 77
  - ars\_prefix\_states, 77
  - ars\_statistics, 77
  - cycle\_states\_, 78
  - inc\_ars\_cycle\_states, 77
  - inc\_ars\_prefix\_states, 78
  - prefix\_states\_, 78
- spot::barand, 84
  - barand, 85
  - m\_, 85
  - n\_, 85
  - rand, 85
  - s\_, 85
- spot::bdd\_allocator, 85
  - allocate\_variables, 89
  - bdd\_allocator, 89
  - dump\_free\_list, 89
  - extend, 89
  - extvarnum, 89
  - fl, 90
  - free\_list\_type, 88
  - initialize, 89
  - initialized, 90
  - insert, 89
  - lvarnum, 90
  - pos\_lenght\_pair, 88
  - register\_n, 89
  - release\_n, 89
  - release\_variables, 90
  - remove, 90
- spot::bdd\_dict, 90
  - ~bdd\_dict, 95
  - acc\_formula\_map, 98
  - acc\_map, 98
  - allocate\_variables, 96
  - assert\_emptiness, 96
  - bdd\_dict, 95
  - dump, 96
  - free\_anonymous\_list\_of, 98
  - free\_anonymous\_list\_of\_type, 95
  - fv\_map, 95
  - initialize, 96

- initialized, 98
- is\_registered\_acceptance\_variable, 96
- is\_registered\_proposition, 96
- is\_registered\_state, 96
- lvarnum, 98
- next\_to\_now, 98
- now\_formula\_map, 98
- now\_map, 98
- now\_to\_next, 98
- operator=, 96
- ref\_set, 95
- register\_acceptance\_variable, 96
- register\_acceptance\_variables, 96
- register\_all\_variables\_of, 97
- register\_anonymous\_variables, 97
- register\_proposition, 97
- register\_propositions, 97
- register\_state, 97
- release\_variables, 97
- unregister\_all\_my\_variables, 98
- unregister\_variable, 98
- var\_formula\_map, 99
- var\_map, 99
- var\_refs, 99
- vf\_map, 95
- vr\_map, 95
- spot::bdd\_dict::annon\_free\_list, 99
  - annon\_free\_list, 101
  - dict\_, 102
  - dump\_free\_list, 101
  - extend, 101
  - fl, 102
  - free\_list\_type, 101
  - insert, 101
  - pos\_lenght\_pair, 101
  - register\_n, 101
  - release\_n, 102
  - remove, 102
- spot::bdd\_less\_than, 102
  - operator(), 103
- spot::bfs\_steps, 103
  - ~bfs\_steps, 104
  - a\_, 105
  - bfs\_steps, 104
  - filter, 104
  - finalize, 104
  - match, 105
  - search, 105
- spot::connected\_component\_hash\_set, 115
  - ~connected\_component\_hash\_set, 117
  - condition, 117
  - has\_state, 117
  - index, 117
  - insert, 117
  - rem, 117
  - set\_type, 117
  - states, 117
- spot::connected\_component\_hash\_set\_factory, 117
  - ~connected\_component\_hash\_set\_factory, 119
  - build, 119
  - connected\_component\_hash\_set\_factory, 119
  - instance, 119
- spot::couvreur99\_check, 124
  - ~couvreur99\_check, 127
  - a\_, 128
  - automaton, 127
  - check, 127
  - couvreur99\_check, 127
  - dec\_depth, 127
  - depth, 127
  - ecs\_, 128
  - inc\_depth, 128
  - inc\_states, 128
  - inc\_transitions, 128
  - max\_depth, 128
  - poprem\_, 128
  - print\_stats, 128
  - remove\_component, 128
  - result, 128
  - set\_states, 128
  - states, 128
  - transitions, 128
- spot::couvreur99\_check\_result, 129
  - a\_, 132
  - accepting\_cycle, 131
  - accepting\_run, 131
  - acss\_states, 131
  - ars\_cycle\_states, 131
  - ars\_prefix\_states, 131
  - automaton, 132
  - couvreur99\_check\_result, 131
  - ecs\_, 132
  - inc\_ars\_cycle\_states, 132
  - inc\_ars\_prefix\_states, 132
  - print\_stats, 132
  - run\_, 132
- spot::couvreur99\_check\_shy, 132
  - ~couvreur99\_check\_shy, 136
  - a\_, 137
  - arc, 137
  - automaton, 136
  - check, 136
  - clear\_todo, 136
  - couvreur99\_check\_shy, 136
  - dec\_depth, 136
  - depth, 136
  - ecs\_, 137

- find\_state, 136
- group\_, 137
- inc\_depth, 136
- inc\_states, 136
- inc\_transitions, 136
- max\_depth, 136
- num, 137
- poprem\_, 137
- print\_stats, 136
- remove\_component, 137
- result, 137
- set\_states, 137
- states, 137
- succ\_queue, 135
- todo, 137
- todo\_list, 135
- transitions, 137
- spot::couvreur99\_check\_shy::successor, 138
  - acc, 138
  - s, 138
  - successor, 138
- spot::couvreur99\_check\_shy::todo\_item, 139
  - n, 139
  - q, 139
  - s, 139
  - todo\_item, 139
- spot::couvreur99\_check\_status, 140
  - ~couvreur99\_check\_status, 141
  - aut, 141
  - couvreur99\_check\_status, 141
  - cycle\_seed, 141
  - h, 141
  - print\_stats, 141
  - root, 141
  - states, 141
- spot::delayed\_simulation\_relation, 146
- spot::direct\_simulation\_relation, 146
- spot::dotty\_decorator, 146
  - ~dotty\_decorator, 147
  - dotty\_decorator, 147
  - instance, 147
  - link\_decl, 147
  - state\_decl, 148
- spot::duplicator\_node, 148
  - ~duplicator\_node, 151
  - acc\_, 152
  - add\_pred, 151
  - add\_succ, 151
  - compare, 151
  - del\_pred, 151
  - del\_succ, 151
  - duplicator\_node, 151
  - get\_acc, 151
  - get\_duplicator\_node, 151
  - get\_label, 151
  - get\_nb\_succ, 151
  - get\_pair, 152
  - get\_spoiler\_node, 152
  - implies, 152
  - label\_, 152
  - lnode\_pred, 152
  - lnode\_succ, 152
  - match, 152
  - not\_win, 152
  - num\_, 152
  - prune, 152
  - sc\_, 152
  - set\_win, 152
  - succ\_to\_string, 152
  - to\_string, 152
- spot::duplicator\_node\_delayed, 153
  - ~duplicator\_node\_delayed, 155
  - acc\_, 157
  - add\_pred, 155
  - add\_succ, 155
  - compare, 156
  - del\_pred, 156
  - del\_succ, 156
  - duplicator\_node\_delayed, 155
  - get\_acc, 156
  - get\_duplicator\_node, 156
  - get\_label, 156
  - get\_lead\_2\_acc\_all, 156
  - get\_nb\_succ, 156
  - get\_pair, 156
  - get\_progress\_measure, 156
  - get\_spoiler\_node, 156
  - implies, 156
  - implies\_acc, 156
  - implies\_label, 156
  - label\_, 157
  - lead\_2\_acc\_all\_, 157
  - lnode\_pred, 157
  - lnode\_succ, 157
  - match, 156
  - not\_win, 157
  - num\_, 157
  - progress\_measure\_, 157
  - prune, 156
  - sc\_, 157
  - seen\_, 157
  - set\_lead\_2\_acc\_all, 156
  - set\_win, 156
  - succ\_to\_string, 157
  - to\_string, 157
- spot::ec\_statistics, 157
  - dec\_depth, 159
  - depth, 159

- depth\_, 159
- ec\_statistics, 159
- inc\_depth, 159
- inc\_states, 159
- inc\_transitions, 159
- max\_depth, 159
- max\_depth\_, 159
- set\_states, 159
- states, 159
- states\_, 159
- transitions, 159
- transitions\_, 160
- spot::emptiness\_check, 160
  - ~emptiness\_check, 162
  - a\_, 162
  - automaton, 162
  - check, 162
  - emptiness\_check, 162
  - print\_stats, 162
- spot::emptiness\_check\_result, 162
  - a\_, 164
  - accepting\_run, 164
  - automaton, 164
  - emptiness\_check\_result, 164
- spot::evtgba, 166
  - ~evtgba, 167
  - all\_acceptance\_conditions, 167
  - alphabet, 167
  - evtgba, 167
  - format\_acceptance\_condition, 167
  - format\_acceptance\_conditions, 167
  - format\_label, 167
  - format\_state, 167
  - init\_iter, 167
  - pred\_iter, 168
  - succ\_iter, 168
- spot::evtgba\_explicit, 168
  - ~evtgba\_explicit, 170
  - acc\_set\_, 172
  - add\_transition, 171
  - all\_acceptance\_conditions, 171
  - alphabet, 171
  - alphabet\_, 172
  - declare\_acceptance\_condition, 171
  - declare\_state, 171
  - evtgba\_explicit, 170
  - format\_acceptance\_condition, 171
  - format\_acceptance\_conditions, 171
  - format\_label, 171
  - format\_state, 171
  - init\_iter, 171
  - init\_states\_, 172
  - name\_state\_map\_, 172
  - ns\_map, 170
  - pred\_iter, 171, 172
  - set\_init\_state, 172
  - sn\_map, 170
  - state\_name\_map\_, 172
  - succ\_iter, 172
  - transition\_list, 170
- spot::evtgba\_explicit::state, 172
  - in, 173
  - out, 173
- spot::evtgba\_explicit::transition, 173
  - acceptance\_conditions, 173
  - in, 173
  - label, 174
  - out, 174
- spot::evtgba\_iterator, 174
  - ~evtgba\_iterator, 174
  - current\_acceptance\_conditions, 174
  - current\_label, 174
  - current\_state, 174
  - done, 174
  - first, 174
  - next, 174
- spot::evtgba\_product, 175
  - ~evtgba\_product, 176
  - all\_acc\_, 177
  - all\_acceptance\_conditions, 177
  - alphabet, 177
  - alphabet\_, 177
  - common\_symbol\_table, 176
  - common\_symbols\_, 178
  - evtgba\_product, 176
  - evtgba\_product\_operands, 176
  - format\_acceptance\_condition, 177
  - format\_acceptance\_conditions, 177
  - format\_label, 177
  - format\_state, 177
  - init\_iter, 177
  - op\_, 178
  - pred\_iter, 177
  - succ\_iter, 177
- spot::evtgba\_reachable\_iterator, 178
  - ~evtgba\_reachable\_iterator, 180
  - add\_state, 180
  - automata\_, 181
  - end, 180
  - evtgba\_reachable\_iterator, 180
  - next\_state, 180
  - process\_link, 180
  - process\_state, 181
  - run, 181
  - seen, 181
  - seen\_map, 180
  - start, 181
- spot::evtgba\_reachable\_iterator\_breadth\_first, 181

- add\_state, 183
- automata\_, 184
- end, 183
- evtgba\_reachable\_iterator\_breadth\_first, 183
- next\_state, 184
- process\_link, 184
- process\_state, 184
- run, 184
- seen, 184
- seen\_map, 183
- start, 184
- todo, 185
- spot::evtgba\_reachable\_iterator\_depth\_first, 185
  - add\_state, 187
  - automata\_, 188
  - end, 187
  - evtgba\_reachable\_iterator\_depth\_first, 187
  - next\_state, 187
  - process\_link, 187
  - process\_state, 188
  - run, 188
  - seen, 188
  - seen\_map, 187
  - start, 188
  - todo, 188
- spot::explicit\_connected\_component, 189
  - ~explicit\_connected\_component, 190
  - condition, 190
  - has\_state, 190
  - index, 190
  - insert, 190
  - rem, 190
- spot::explicit\_connected\_component\_factory, 191
  - ~explicit\_connected\_component\_factory, 191
  - build, 191
- spot::free\_list, 196
  - ~free\_list, 198
  - dump\_free\_list, 198
  - extend, 198
  - fl, 199
  - free\_list\_type, 198
  - insert, 199
  - pos\_lenght\_pair, 198
  - register\_n, 199
  - release\_n, 199
  - remove, 199
- spot::gspn\_exeption, 199
  - err\_, 200
  - get\_err, 200
  - get\_where, 200
  - gspn\_exeption, 200
  - where\_, 200
- spot::gspn\_interface, 200
  - ~gspn\_interface, 201
- automaton, 202
- dead\_, 202
- dict\_, 202
- env\_, 202
- gspn\_interface, 201
- spot::gspn\_ssp\_interface, 202
  - ~gspn\_ssp\_interface, 203
- automaton, 204
- dict\_, 204
- env\_, 204
- gspn\_ssp\_interface, 203
- spot::loopless\_modular\_mixed\_radix\_gray\_code, 206
  - ~loopless\_modular\_mixed\_radix\_gray\_code, 207
- a\_, 208
- a\_first, 208
- a\_last, 208
- a\_next, 208
- done, 208
- done\_, 208
- f\_, 208
- first, 208
- last, 208
- loopless\_modular\_mixed\_radix\_gray\_code, 207
- m\_, 208
- n\_, 208
- next, 208
- non\_one\_radixes\_, 209
- s\_, 209
- spot::ltl, 70
  - unabbreviate\_ltl, 74
- spot::ltl::atomic\_prop, 78
  - ~atomic\_prop, 82
  - accept, 82
  - atomic\_prop, 82
  - dump, 82
  - dump\_, 84
  - dump\_instances, 82
  - env, 82
  - env\_, 84
  - hash, 82
  - hash\_key\_, 84
  - instance, 83
  - instance\_count, 83
  - instances, 84
  - map, 82
  - name, 83
  - name\_, 84
  - pair, 82
  - ref, 83
  - ref\_, 83
  - ref\_count\_, 83

- set\_key\_, 83
  - unref, 83
  - unref\_, 83
- spot::ltl::binop, 105
  - ~binop, 110
  - accept, 110
  - binop, 110
  - dump, 110
  - dump\_, 112
  - Equiv, 110
  - first, 110
  - first\_, 112
  - hash, 110
  - hash\_key\_, 112
  - Implies, 109
  - instance, 110
  - instance\_count, 110
  - instances, 112
  - map, 109
  - op, 110
  - op\_, 112
  - op\_name, 111
  - pair, 109
  - pairf, 109
  - R, 110
  - ref, 111
  - ref\_, 111
  - ref\_count\_, 111
  - second, 111
  - second\_, 112
  - set\_key\_, 111
  - type, 109
  - U, 110
  - unref, 111
  - unref\_, 111
  - Xor, 109
- spot::ltl::clone\_visitor, 112
  - ~clone\_visitor, 114
  - clone\_visitor, 114
  - recurse, 114
  - result, 114
  - result\_, 115
  - visit, 114, 115
- spot::ltl::const\_visitor, 119
  - visit, 120
- spot::ltl::constant, 120
  - ~constant, 123
  - accept, 123
  - constant, 123
  - dump, 123
  - dump\_, 124
  - False, 122
  - false\_instance, 123
  - hash, 123
  - hash\_key\_, 124
  - ref, 123
  - ref\_, 123
  - set\_key\_, 123
  - True, 122
  - true\_instance, 123
  - type, 122
  - unref, 123
  - unref\_, 124
  - val, 124
  - val\_, 124
  - val\_name, 124
- spot::ltl::declarative\_environment, 141
  - ~declarative\_environment, 143
  - declarative\_environment, 143
  - declare, 143
  - get\_prop\_map, 143
  - name, 143
  - prop\_map, 143
  - props\_, 144
  - require, 143
- spot::ltl::default\_environment, 144
  - ~default\_environment, 145
  - default\_environment, 145
  - instance, 145
  - name, 145
  - require, 145
- spot::ltl::environment, 164
  - ~environment, 165
  - name, 165
  - require, 165
- spot::ltl::formula, 191
  - ~formula, 193
  - accept, 193
  - dump, 194
  - dump\_, 194
  - hash, 194
  - hash\_key\_, 194
  - ref, 194
  - ref\_, 194
  - set\_key\_, 194
  - unref, 194
  - unref\_, 194
- spot::ltl::formula\_ptr\_hash, 195
  - operator(), 195
- spot::ltl::formula\_ptr\_less\_than, 195
  - operator(), 196
- spot::ltl::multop, 212
  - ~multop, 217
  - accept, 217
  - And, 217
  - children\_, 219
  - dump, 217
  - dump\_, 219

- hash, 217
- hash\_key\_, 219
- instance, 217
- instance\_count, 218
- instances, 219
- map, 216
- multop, 217
- nth, 218
- op, 218
- op\_, 219
- op\_name, 218
- Or, 217
- pair, 216
- ref, 218
- ref\_, 218
- ref\_count\_, 218
- set\_key\_, 218
- size, 219
- type, 217
- unref, 219
- unref\_, 219
- vec, 216
- spot::ltl::multop::pairemp, 219
  - operator(), 220
- spot::ltl::postfix\_visitor, 249
  - ~postfix\_visitor, 250
  - doit, 250
  - doit\_default, 250
  - postfix\_visitor, 250
  - visit, 250, 251
- spot::ltl::random\_ltl, 251
  - ~random\_ltl, 253
  - ap, 253
  - ap\_, 254
  - dump\_priorities, 253
  - generate, 253
  - parse\_options, 253
  - proba\_, 254
  - proba\_2\_, 254
  - random\_ltl, 253
  - total\_1\_, 254
  - total\_2\_, 254
  - total\_2\_and\_more\_, 254
  - update\_sums, 254
- spot::ltl::random\_ltl::op\_proba, 254
  - build, 255
  - builder, 255
  - min\_n, 255
  - name, 255
  - proba, 255
  - setup, 255
- spot::ltl::ref\_formula, 255
  - ~ref\_formula, 258
  - accept, 258
  - dump, 258
  - dump\_, 259
  - hash, 258
  - hash\_key\_, 259
  - ref, 258
  - ref\_, 258
  - ref\_count\_, 258
  - ref\_counter\_, 259
  - ref\_formula, 258
  - set\_key\_, 259
  - unref, 259
  - unref\_, 259
- spot::ltl::simplify\_f\_g\_visitor, 264
  - ~simplify\_f\_g\_visitor, 266
  - recurse, 266
  - result, 266
  - result\_, 266
  - simplify\_f\_g\_visitor, 266
  - super, 266
  - visit, 266
- spot::ltl::unabbreviate\_logic\_visitor, 395
  - ~unabbreviate\_logic\_visitor, 398
  - recurse, 398
  - result, 398
  - result\_, 399
  - super, 398
  - unabbreviate\_logic\_visitor, 398
  - visit, 398
- spot::ltl::unabbreviate\_ltl\_visitor, 399
  - ~unabbreviate\_ltl\_visitor, 401
  - recurse, 401
  - result, 401
  - result\_, 402
  - super, 401
  - unabbreviate\_ltl\_visitor, 401
  - visit, 401, 402
- spot::ltl::unop, 402
  - ~unop, 406
  - accept, 407
  - child, 407
  - child\_, 408
  - dump, 407
  - dump\_, 408
  - F, 406
  - G, 406
  - hash, 407
  - hash\_key\_, 408
  - instance, 407
  - instance\_count, 407
  - instances, 408
  - map, 406
  - Not, 406
  - op, 407
  - op\_, 408

- op\_name, 407
- pair, 406
- ref, 407
- ref\_, 407
- ref\_count\_, 408
- set\_key\_, 408
- type, 406
- unop, 406
- unref, 408
- unref\_, 408
- X, 406
- spot::ltl::visitor, 409
  - visit, 410
- spot::minato\_isop, 209
  - cube\_, 210
  - minato\_isop, 210
  - next, 210
  - ret\_, 210
  - todo\_, 210
- spot::minato\_isop::local\_vars
  - FirstStep, 211
  - FourthStep, 211
  - SecondStep, 211
  - ThirdStep, 211
- spot::minato\_isop::local\_vars, 211
  - f0\_max, 211
  - f0\_min, 211
  - f1\_max, 212
  - f1\_min, 212
  - f\_max, 212
  - f\_min, 212
  - g0, 212
  - g1, 212
  - local\_vars, 211
  - step, 212
  - v1, 212
  - vars, 212
- spot::numbered\_state\_heap, 220
  - ~numbered\_state\_heap, 221
  - find, 221
  - index, 222
  - index\_and\_insert, 222
  - insert, 222
  - iterator, 222
  - size, 222
  - state\_index, 221
  - state\_index\_p, 221
- spot::numbered\_state\_heap\_const\_iterator, 223
  - ~numbered\_state\_heap\_const\_iterator, 223
  - done, 223
  - first, 223
  - get\_index, 223
  - get\_state, 223
  - next, 223
- spot::numbered\_state\_heap\_factory, 224
  - ~numbered\_state\_heap\_factory, 224
  - build, 224
- spot::numbered\_state\_heap\_hash\_map, 225
  - ~numbered\_state\_heap\_hash\_map, 227
  - find, 227
  - h, 228
  - hash\_type, 226
  - index, 227
  - index\_and\_insert, 227
  - insert, 227
  - iterator, 228
  - size, 228
  - state\_index, 226
  - state\_index\_p, 226
- spot::numbered\_state\_heap\_hash\_map\_factory, 228
  - ~numbered\_state\_heap\_hash\_map\_factory, 229
  - build, 229
  - instance, 229
  - numbered\_state\_heap\_hash\_map\_factory, 229
- spot::parity\_game\_graph, 230
  - ~parity\_game\_graph, 232
  - add\_state, 233
  - automata\_, 234
  - build\_graph, 233
  - duplicator\_vertice\_, 234
  - end, 233
  - get\_relation, 233
  - lift, 233
  - nb\_node\_parity\_game, 234
  - next\_state, 233
  - parity\_game\_graph, 232
  - print, 233
  - process\_link, 233
  - process\_state, 234
  - run, 234
  - seen, 234
  - seen\_map, 232
  - spoiler\_vertice\_, 234
  - start, 234
  - tgba\_state\_, 234
  - todo, 234
- spot::parity\_game\_graph\_delayed, 235
  - ~parity\_game\_graph\_delayed, 238
  - add\_duplicator\_node\_delayed, 238
  - add\_spoiler\_node\_delayed, 238
  - add\_state, 239
  - automata\_, 240
  - bdd\_v, 238
  - build\_graph, 239
  - build\_recurse\_successor\_duplicator, 239
  - build\_recurse\_successor\_spoiler, 239



- duplicator\_vertice\_, 240
- end, 239
- get\_relation, 239
- lift, 239
- nb\_node\_parity\_game, 240
- nb\_set\_acc\_cond, 239
- next\_state, 239
- parity\_game\_graph\_delayed, 238
- print, 239
- process\_link, 239, 240
- process\_state, 240
- run, 240
- seen, 241
- seen\_map, 238
- spoiler\_vertice\_, 241
- start, 240
- sub\_set\_acc\_cond\_, 241
- tgba\_state\_, 241
- todo, 241
- spot::parity\_game\_graph\_direct, 241
  - ~parity\_game\_graph\_direct, 245
  - add\_state, 245
  - automata\_, 246
  - build\_graph, 245
  - build\_link, 245
  - duplicator\_vertice\_, 246
  - end, 245
  - get\_relation, 245
  - lift, 245
  - nb\_node\_parity\_game, 246
  - next\_state, 245
  - parity\_game\_graph\_direct, 245
  - print, 245
  - process\_link, 245, 246
  - process\_state, 246
  - run, 246
  - seen, 246
  - seen\_map, 244
  - spoiler\_vertice\_, 247
  - start, 246
  - tgba\_state\_, 247
  - todo, 247
- spot::ptr\_hash, 251
  - operator(), 251
- spot::rsymbol, 259
  - ~rsymbol, 260
  - operator const symbol \*, 261
  - operator!=, 261
  - operator<, 261
  - operator=, 261
  - operator==, 261
  - rsymbol, 260
  - s\_, 261
- spot::scc\_stack, 261
  - clear\_rem, 262
  - empty, 262
  - pop, 262
  - push, 262
  - rem, 262
  - s, 263
  - size, 262
  - stack\_type, 262
  - top, 262, 263
- spot::scc\_stack::connected\_component, 263
  - condition, 264
  - connected\_component, 264
  - index, 264
  - rem, 264
- spot::spoiler\_node, 268
  - ~spoiler\_node, 269
  - add\_pred, 269
  - add\_succ, 269
  - compare, 269
  - del\_pred, 269
  - del\_succ, 269
  - get\_duplicator\_node, 269
  - get\_nb\_succ, 270
  - get\_pair, 270
  - get\_spoiler\_node, 270
  - lnode\_pred, 270
  - lnode\_succ, 270
  - not\_win, 270
  - num\_, 270
  - prune, 270
  - sc\_, 270
  - set\_win, 270
  - spoiler\_node, 269
  - succ\_to\_string, 270
  - to\_string, 270
- spot::spoiler\_node\_delayed, 270
  - ~spoiler\_node\_delayed, 272
  - acceptance\_condition\_visited\_, 274
  - add\_pred, 273
  - add\_succ, 273
  - compare, 273
  - del\_pred, 273
  - del\_succ, 273
  - get\_acceptance\_condition\_visited, 273
  - get\_duplicator\_node, 273
  - get\_lead\_2\_acc\_all, 273
  - get\_nb\_succ, 273
  - get\_pair, 273
  - get\_progress\_measure, 273
  - get\_spoiler\_node, 273
  - lead\_2\_acc\_all\_, 274
  - lnode\_pred, 274
  - lnode\_succ, 274
  - not\_win, 274

- num\_, 274
- progress\_measure\_, 274
- prune, 273
- sc\_, 274
- seen\_, 274
- set\_lead\_2\_acc\_all, 273
- set\_win, 273
- spoiler\_node\_delayed, 272
- succ\_to\_string, 273
- to\_string, 273
- spot::state, 276
  - ~state, 277
  - clone, 277
  - compare, 277
  - hash, 278
- spot::state\_bdd, 278
  - as\_bdd, 280
  - clone, 280
  - compare, 280
  - hash, 280
  - state\_, 280
  - state\_bdd, 279
- spot::state\_evtgba\_explicit, 280
  - ~state\_evtgba\_explicit, 282
  - clone, 282
  - compare, 282
  - get\_state, 282
  - hash, 282
  - state\_, 283
  - state\_evtgba\_explicit, 282
- spot::state\_explicit, 283
  - ~state\_explicit, 284
  - clone, 285
  - compare, 285
  - get\_state, 285
  - hash, 285
  - state\_, 285
  - state\_explicit, 284
- spot::state\_product, 286
  - ~state\_product, 287
  - clone, 288
  - compare, 288
  - hash, 288
  - left, 288
  - left\_, 288
  - right, 288
  - right\_, 288
  - state\_product, 287
- spot::state\_ptr\_equal, 289
  - operator(), 289
- spot::state\_ptr\_hash, 289
  - operator(), 290
- spot::state\_ptr\_less\_than, 290
  - operator(), 290
- spot::string\_hash, 290
  - operator(), 291
- spot::symbol, 291
  - ~symbol, 292
  - dump\_instances, 292
  - instance, 292
  - instance\_count, 292
  - instances\_, 292
  - map, 292
  - name, 292
  - name\_, 292
  - ref, 292
  - ref\_count\_, 292
  - refs\_, 292
  - symbol, 292
  - unref, 292
- spot::tgba, 293
  - ~tgba, 295
  - all\_acceptance\_conditions, 296
  - compute\_support\_conditions, 296
  - compute\_support\_variables, 296
  - format\_state, 296
  - get\_dict, 296
  - get\_init\_state, 296
  - last\_support\_conditions\_input\_, 298
  - last\_support\_conditions\_output\_, 298
  - last\_support\_variables\_input\_, 298
  - last\_support\_variables\_output\_, 298
  - neg\_acceptance\_conditions, 296
  - num\_acc\_, 298
  - number\_of\_acceptance\_conditions, 297
  - project\_state, 297
  - succ\_iter, 297
  - support\_conditions, 297
  - support\_variables, 298
  - tgba, 295
  - transition\_annotation, 298
- spot::tgba\_bdd\_concrete, 299
  - ~tgba\_bdd\_concrete, 302
  - all\_acceptance\_conditions, 302
  - compute\_support\_conditions, 302
  - compute\_support\_variables, 302
  - data\_, 305
  - format\_state, 302
  - get\_core\_data, 303
  - get\_dict, 303
  - get\_init\_bdd, 303
  - get\_init\_state, 303
  - init\_, 305
  - neg\_acceptance\_conditions, 303
  - number\_of\_acceptance\_conditions, 303
  - project\_state, 304
  - set\_init\_state, 304
  - succ\_iter, 304

- support\_conditions, 304
- support\_variables, 305
- tgba\_bdd\_concrete, 302
- tgba\_bdd\_concrete::operator=, 305
- transition\_annotation, 305
- spot::tgba\_bdd\_concrete\_factory, 305
  - ~tgba\_bdd\_concrete\_factory, 307
  - acc\_, 309
  - acc\_map\_, 307
  - constrain\_relation, 307
  - create\_atomic\_prop, 307
  - create\_state, 308
  - data\_, 309
  - declare\_acceptance\_condition, 308
  - finish, 308
  - get\_core\_data, 308
  - get\_dict, 308
  - tgba\_bdd\_concrete\_factory, 307
- spot::tgba\_bdd\_core\_data, 309
  - acc\_set, 312
  - acceptance\_conditions, 312
  - all\_acceptance\_conditions, 313
  - declare\_acceptance\_condition, 312
  - declare\_atomic\_prop, 312
  - declare\_now\_next, 312
  - dict, 313
  - negacc\_set, 313
  - next\_set, 313
  - notacc\_set, 313
  - notnext\_set, 313
  - notnow\_set, 313
  - notvar\_set, 314
  - now\_set, 314
  - nownext\_set, 314
  - operator=, 312
  - relation, 314
  - tgba\_bdd\_core\_data, 312
  - var\_set, 314
  - varandnext\_set, 314
- spot::tgba\_bdd\_factory, 314
  - get\_core\_data, 315
- spot::tgba\_explicit, 315
  - ~tgba\_explicit, 320
  - add\_acceptance\_condition, 320
  - add\_acceptance\_conditions, 320
  - add\_condition, 320
  - add\_conditions, 320
  - add\_state, 320
  - all\_acceptance\_conditions, 320
  - all\_acceptance\_conditions\_, 324
  - all\_acceptance\_conditions\_computed\_, 324
  - complement\_all\_acceptance\_conditions, 320
  - compute\_support\_conditions, 320, 321
  - compute\_support\_variables, 321
  - copy\_acceptance\_conditions\_of, 321
  - create\_transition, 321
  - declare\_acceptance\_condition, 321
  - dict\_, 324
  - format\_state, 321
  - get\_acceptance\_condition, 321
  - get\_dict, 321
  - get\_init\_state, 322
  - has\_acceptance\_condition, 322
  - init\_, 324
  - merge\_transitions, 322
  - name\_state\_map\_, 324
  - neg\_acceptance\_conditions, 322
  - neg\_acceptance\_conditions\_, 324
  - ns\_map, 320
  - number\_of\_acceptance\_conditions, 322
  - project\_state, 322
  - set\_init\_state, 322
  - sn\_map, 320
  - state, 320
  - state\_name\_map\_, 324
  - succ\_iter, 323
  - support\_conditions, 323
  - support\_variables, 323
  - tgba\_explicit, 320
  - tgba\_explicit::operator=, 323
  - transition\_annotation, 323
- spot::tgba\_explicit::transition, 324
  - acceptance\_conditions, 325
  - condition, 325
  - dest, 325
- spot::tgba\_explicit\_succ\_iterator, 325
  - ~tgba\_explicit\_succ\_iterator, 327
  - all\_acceptance\_conditions\_, 328
  - current\_acceptance\_conditions, 327
  - current\_condition, 327
  - current\_state, 327
  - done, 327
  - first, 327
  - i\_, 328
  - next, 328
  - s\_, 328
  - tgba\_explicit\_succ\_iterator, 327
- spot::tgba\_product, 328
  - ~tgba\_product, 332
  - all\_acceptance\_conditions, 332
  - all\_acceptance\_conditions\_, 335
  - compute\_support\_conditions, 332
  - compute\_support\_variables, 332
  - dict\_, 335
  - format\_state, 332
  - get\_dict, 332
  - get\_init\_state, 333
  - left\_, 335

- left\_acc\_complement\_, 335
- neg\_acceptance\_conditions, 333
- neg\_acceptance\_conditions\_, 335
- number\_of\_acceptance\_conditions, 333
- project\_state, 333
- right\_, 335
- right\_acc\_complement\_, 335
- succ\_iter, 333
- support\_conditions, 334
- support\_variables, 334
- tgba\_product, 332
- tgba\_product::operator=, 334
- transition\_annotation, 334
- spot::tgba\_reachable\_iterator, 335
  - ~tgba\_reachable\_iterator, 337
  - add\_state, 337
  - automata\_, 338
  - end, 337
  - next\_state, 337
  - process\_link, 338
  - process\_state, 338
  - run, 338
  - seen, 338
  - seen\_map, 337
  - start, 338
  - tgba\_reachable\_iterator, 337
- spot::tgba\_reachable\_iterator\_breadth\_first, 339
  - add\_state, 341
  - automata\_, 343
  - end, 341
  - next\_state, 342
  - process\_link, 342
  - process\_state, 342
  - run, 342
  - seen, 343
  - seen\_map, 341
  - start, 342
  - tgba\_reachable\_iterator\_breadth\_first, 341
  - todo, 343
- spot::tgba\_reachable\_iterator\_depth\_first, 343
  - add\_state, 345
  - automata\_, 346
  - end, 345
  - next\_state, 345
  - process\_link, 346
  - process\_state, 346
  - run, 346
  - seen, 346
  - seen\_map, 345
  - start, 346
  - tgba\_reachable\_iterator\_depth\_first, 345
  - todo, 347
- spot::tgba\_reduc, 347
  - ~tgba\_reduc, 353
- acc\_, 361
- add\_acceptance\_condition, 353
- add\_acceptance\_conditions, 353
- add\_condition, 354
- add\_conditions, 354
- add\_state, 354
- all\_acceptance\_conditions, 354
- all\_acceptance\_conditions\_, 361
- all\_acceptance\_conditions\_computed\_, 361
- automata\_, 361
- complement\_all\_acceptance\_conditions, 354
- compute\_scc, 354
- compute\_support\_conditions, 354
- compute\_support\_variables, 354
- copy\_acceptance\_conditions\_of, 355
- create\_transition, 355
- declare\_acceptance\_condition, 355
- delete\_scc, 355
- delete\_transitions, 355
- dict\_, 361
- display\_rel\_sim, 355
- display\_scc, 355
- end, 355
- format\_state, 355
- get\_acceptance\_condition, 356
- get\_dict, 356
- get\_init\_state, 356
- h\_, 361
- has\_acceptance\_condition, 356
- init\_, 361
- is\_not\_accepting, 356
- is\_terminal, 356
- merge\_state, 356
- merge\_state\_delayed, 357
- merge\_transitions, 357
- name\_state\_map\_, 361
- neg\_acceptance\_conditions, 357
- neg\_acceptance\_conditions\_, 361
- next\_state, 357
- ns\_map, 353
- number\_of\_acceptance\_conditions, 357
- process\_link, 357
- process\_state, 358
- project\_state, 358
- prune\_acc, 358
- prune\_scc, 358
- quotient\_state, 358
- redirect\_transition, 358
- remove\_acc, 359
- remove\_component, 359
- remove\_predecessor\_state, 359
- remove\_scc, 359
- remove\_state, 359
- root\_, 361

- run, 359
- scc\_computed\_, 361
- seen, 361
- seen\_, 361
- seen\_map, 353
- set\_init\_state, 359
- si\_, 361
- sn\_map, 353
- sp\_map, 353
- start, 359
- state, 353
- state\_name\_map\_, 361
- state\_predecessor\_map\_, 361
- state\_scc\_, 361
- state\_scc\_v\_, 361
- succ\_iter, 359, 360
- support\_conditions, 360
- support\_variables, 360
- tgba\_reduc, 353
- tgba\_reduc::nb\_set\_acc\_cond, 360
- todo, 362
- transition\_annotation, 360
- spot::tgba\_run, 362
  - ~tgba\_run, 362
  - cycle, 363
  - operator=, 363
  - prefix, 363
  - steps, 362
  - tgba\_run, 362, 363
- spot::tgba\_run::step, 363
  - acc, 363
  - label, 363
  - s, 364
- spot::tgba\_run\_dotty\_decorator, 364
  - ~tgba\_run\_dotty\_decorator, 365
  - instance, 366
  - link\_decl, 366
  - map\_, 366
  - run\_, 366
  - state\_decl, 366
  - step\_map, 365
  - step\_num, 365
  - step\_set, 365
  - tgba\_run\_dotty\_decorator, 365
- spot::tgba\_sba\_proxy, 367
  - acc\_cycle\_, 373
  - all\_acceptance\_conditions, 370
  - compute\_support\_conditions, 370
  - compute\_support\_variables, 370
  - cycle\_list, 370
  - format\_state, 370
  - get\_dict, 370
  - get\_init\_state, 370
  - neg\_acceptance\_conditions, 371
  - number\_of\_acceptance\_conditions, 371
  - project\_state, 371
  - state\_is\_accepting, 371
  - succ\_iter, 371
  - support\_conditions, 372
  - support\_variables, 372
  - tgba\_sba\_proxy, 370
  - transition\_annotation, 372
- spot::tgba\_statistics, 373
  - states, 373
  - transitions, 373
- spot::tgba\_succ\_iterator, 373
  - ~tgba\_succ\_iterator, 375
  - current\_acceptance\_conditions, 375
  - current\_condition, 375
  - current\_state, 375
  - done, 375
  - first, 375
  - next, 376
- spot::tgba\_succ\_iterator\_concrete, 376
  - ~tgba\_succ\_iterator\_concrete, 378
  - current\_, 379
  - current\_acc\_, 379
  - current\_acceptance\_conditions, 378
  - current\_condition, 378
  - current\_state, 379
  - current\_state\_, 380
  - data\_, 380
  - done, 379
  - first, 379
  - next, 379
  - succ\_set\_, 380
  - succ\_set\_left\_, 380
  - tgba\_succ\_iterator\_concrete, 378
- spot::tgba\_succ\_iterator\_product, 380
  - ~tgba\_succ\_iterator\_product, 383
  - current\_acceptance\_conditions, 383
  - current\_cond\_, 384
  - current\_condition, 383
  - current\_state, 383
  - done, 383
  - first, 383
  - left\_, 384
  - left\_neg\_, 384
  - next, 384
  - next\_non\_false\_, 384
  - right\_, 384
  - right\_neg\_, 384
  - step\_, 384
  - tgba\_product, 384
  - tgba\_succ\_iterator\_product, 383
- spot::tgba\_tba\_proxy, 384
  - ~tgba\_tba\_proxy, 388
  - a\_, 391

- acc\_cycle\_, 391
- all\_acceptance\_conditions, 388
- compute\_support\_conditions, 388
- compute\_support\_variables, 388
- cycle\_list, 388
- format\_state, 388
- get\_dict, 389
- get\_init\_state, 389
- neg\_acceptance\_conditions, 389
- number\_of\_acceptance\_conditions, 389
- project\_state, 389
- succ\_iter, 389
- support\_conditions, 390
- support\_variables, 390
- tgba\_tba\_proxy, 388
- tgba\_tba\_proxy::operator=, 390
- the\_acceptance\_cond\_, 391
- transition\_annotation, 390
- spot::time\_info, 391
  - stime, 391
  - time\_info, 391
  - utime, 391
- spot::timer, 392
  - start, 393
  - start\_, 393
  - stime, 393
  - stop, 393
  - total\_, 393
  - utime, 393
- spot::timer\_map, 393
  - cancel, 394
  - empty, 394
  - item\_type, 394
  - print, 394
  - start, 395
  - stop, 395
  - timer, 395
  - tm, 395
  - tm\_type, 394
- spot::weight, 410
  - dec\_weight\_handler, 411
  - inc\_weight\_handler, 411
  - m, 412
  - neg\_all\_acc, 412
  - operator+=, 411
  - operator-, 411
  - operator=, 411
  - operator<=, 412
  - pm, 412
  - weight, 411
  - weight\_vector, 411
- srand
  - random, 30
- Stack
  - yy::Stack, 276
- stack\_
  - yy::Slice, 267
- stack\_type
  - spot::scc\_stack, 262
- start
  - spot::evtgba\_reachable\_iterator, 181
  - spot::evtgba\_reachable\_iterator\_breadth\_first, 184
  - spot::evtgba\_reachable\_iterator\_depth\_first, 188
  - spot::parity\_game\_graph, 234
  - spot::parity\_game\_graph\_delayed, 240
  - spot::parity\_game\_graph\_direct, 246
  - spot::tgba\_reachable\_iterator, 338
  - spot::tgba\_reachable\_iterator\_breadth\_first, 342
  - spot::tgba\_reachable\_iterator\_depth\_first, 346
  - spot::tgba\_reduc, 359
  - spot::timer, 393
  - spot::timer\_map, 395
- start\_
  - spot::timer, 393
- state
  - spot::tgba\_explicit, 320
  - spot::tgba\_reduc, 353
- state\_
  - spot::state\_bdd, 280
  - spot::state\_evtgba\_explicit, 283
  - spot::state\_explicit, 285
- state\_bdd
  - spot::state\_bdd, 279
- state\_couple
  - spot, 66
  - tgbareduc.hh, 472
- state\_decl
  - spot::dotty\_decorator, 148
  - spot::tgba\_run\_dotty\_decorator, 366
- state\_evtgba\_explicit
  - spot::state\_evtgba\_explicit, 282
- state\_explicit
  - spot::state\_explicit, 284
- state\_index
  - spot::numbered\_state\_heap, 221
  - spot::numbered\_state\_heap\_hash\_map, 226
- state\_index\_p
  - spot::numbered\_state\_heap, 221
  - spot::numbered\_state\_heap\_hash\_map, 226
- state\_is\_accepting
  - spot::tgba\_sba\_proxy, 371
- state\_name\_map\_
  - spot::evtgba\_explicit, 172
  - spot::tgba\_explicit, 324
  - spot::tgba\_reduc, 361

- state\_predecessor\_map\_
  - spot::tgba\_reduc, 361
- state\_product
  - spot::state\_product, 287
- state\_scc\_
  - spot::tgba\_reduc, 361
- state\_scc\_v\_
  - spot::tgba\_reduc, 361
- states
  - spot::connected\_component\_hash\_set, 117
  - spot::couvreur99\_check, 128
  - spot::couvreur99\_check\_shy, 137
  - spot::couvreur99\_check\_status, 141
  - spot::ec\_statistics, 159
  - spot::tgba\_statistics, 373
- states\_
  - spot::ec\_statistics, 159
- stats\_reachable
  - tgba\_misc, 40
- step
  - spot::minato\_isop::local\_vars, 212
  - yy::Location, 205
- step\_
  - spot::tgba\_succ\_iterator\_product, 384
- step\_map
  - spot::tgba\_run\_dotty\_decorator, 365
- step\_num
  - spot::tgba\_run\_dotty\_decorator, 365
- step\_set
  - spot::tgba\_run\_dotty\_decorator, 365
- steps
  - spot::tgba\_run, 362
- stime
  - spot::time\_info, 391
  - spot::timer, 393
- stop
  - spot::timer, 393
  - spot::timer\_map, 395
- sub\_set\_acc\_cond\_
  - spot::parity\_game\_graph\_delayed, 241
- succ\_iter
  - spot::evtgba, 168
  - spot::evtgba\_explicit, 172
  - spot::evtgba\_product, 177
  - spot::tgba, 297
  - spot::tgba\_bdd\_concrete, 304
  - spot::tgba\_explicit, 323
  - spot::tgba\_product, 333
  - spot::tgba\_reduc, 359, 360
  - spot::tgba\_sba\_proxy, 371
  - spot::tgba\_tba\_proxy, 389
- succ\_queue
  - spot::couvreur99\_check\_shy, 135
- succ\_set\_
  - spot::tgba\_succ\_iterator\_concrete, 380
- succ\_set\_left\_
  - spot::tgba\_succ\_iterator\_concrete, 380
- succ\_to\_string
  - spot::duplicator\_node, 152
  - spot::duplicator\_node\_delayed, 157
  - spot::spoilier\_node, 270
  - spot::spoilier\_node\_delayed, 273
- successor
  - spot::couvreur99\_check\_shy::successor, 138
- super
  - spot::ltl::simplify\_f\_g\_visitor, 266
  - spot::ltl::unabbreviate\_logic\_visitor, 398
  - spot::ltl::unabbreviate\_ltl\_visitor, 401
- support\_conditions
  - spot::tgba, 297
  - spot::tgba\_bdd\_concrete, 304
  - spot::tgba\_explicit, 323
  - spot::tgba\_product, 334
  - spot::tgba\_reduc, 360
  - spot::tgba\_sba\_proxy, 372
  - spot::tgba\_tba\_proxy, 390
- support\_variables
  - spot::tgba, 298
  - spot::tgba\_bdd\_concrete, 305
  - spot::tgba\_explicit, 323
  - spot::tgba\_product, 334
  - spot::tgba\_reduc, 360
  - spot::tgba\_sba\_proxy, 372
  - spot::tgba\_tba\_proxy, 390
- symbol
  - spot::symbol, 292
- symbol.hh
  - rsymbol\_set, 418
  - symbol\_set, 418
- symbol\_set
  - spot, 66
  - symbol.hh, 418
- syntactic\_implication
  - ltl\_misc, 26
- syntactic\_implication\_neg
  - ltl\_misc, 26
- tgba
  - spot::tgba, 295
- TGBA (Transition-based Generalized Büchi Automata), 17
- TGBA algorithms, 32
- TGBA on-the-fly algorithms, 32
- TGBA representations, 31
- TGBA runs and supporting functions, 48
- TGBA simplifications, 37
- tgba/ Directory Reference, 55
- tgba/bdddict.hh, 455

- tgba/bddprint.hh, 456
- tgba/formula2bdd.hh, 460
- tgba/public.hh, 427
- tgba/state.hh, 461
- tgba/statebdd.hh, 462
- tgba/succiter.hh, 462
- tgba/succiterconcrete.hh, 463
- tgba/tgba.hh, 464
- tgba/tgababddconcrete.hh, 466
- tgba/tgababddconcretefactory.hh, 466
- tgba/tgababddconcreteproduct.hh, 467
- tgba/tgababddcoredata.hh, 468
- tgba/tgababddfactory.hh, 468
- tgba/tgbaexplicit.hh, 469
- tgba/tgbaproduct.hh, 470
- tgba/tgbareduc.hh, 471
- tgba/tgbatba.hh, 472
- tgba2evtgba.hh
  - tgba\_to\_evtgba, 423
- tgba\_algorithms
  - product, 32
- tgba\_bdd\_concrete
  - spot::tgba\_bdd\_concrete, 302
- tgba\_bdd\_concrete::operator=
  - spot::tgba\_bdd\_concrete, 305
- tgba\_bdd\_concrete\_factory
  - spot::tgba\_bdd\_concrete\_factory, 307
- tgba\_bdd\_core\_data
  - spot::tgba\_bdd\_core\_data, 312
- tgba\_dupexp\_bfs
  - tgba\_misc, 40
- tgba\_dupexp\_dfs
  - tgba\_misc, 40
- tgba\_explicit
  - spot::tgba\_explicit, 320
- tgba\_explicit::operator=
  - spot::tgba\_explicit, 323
- tgba\_explicit\_succ\_iterator
  - spot::tgba\_explicit\_succ\_iterator, 327
- tgba\_io
  - dotty\_reachable, 33
  - format\_tgba\_parse\_errors, 33
  - lbtt\_reachable, 34
  - never\_claim\_reachable, 34
  - tgba\_parse, 34
  - tgba\_parse\_error, 33
  - tgba\_parse\_error\_list, 33
  - tgba\_save\_reachable, 34
- tgba\_ltl
  - ltl\_to\_tgba\_fm, 35
  - ltl\_to\_tgba\_lacim, 36
- tgba\_misc
  - random\_graph, 40
  - stats\_reachable, 40
- tgba\_dupexp\_bfs, 40
- tgba\_dupexp\_dfs, 40
- tgba\_powerset, 40
- tgba\_parse
  - tgba\_io, 34
- tgba\_parse\_error
  - tgba\_io, 33
- tgba\_parse\_error\_list
  - tgba\_io, 33
- tgba\_powerset
  - tgba\_misc, 40
- tgba\_product
  - spot::tgba\_product, 332
  - spot::tgba\_succ\_iterator\_product, 384
- tgba\_product::operator=
  - spot::tgba\_product, 334
- tgba\_reachable\_iterator
  - spot::tgba\_reachable\_iterator, 337
- tgba\_reachable\_iterator\_breadth\_first
  - spot::tgba\_reachable\_iterator\_breadth\_first, 341
- tgba\_reachable\_iterator\_depth\_first
  - spot::tgba\_reachable\_iterator\_depth\_first, 345
- tgba\_reduc
  - spot::tgba\_reduc, 353
- tgba\_reduc::nb\_set\_acc\_cond
  - spot::tgba\_reduc, 360
- tgba\_reduction
  - Reduce\_All, 38
  - Reduce\_None, 38
  - Reduce\_quotient\_Del\_Sim, 38
  - Reduce\_quotient\_Dir\_Sim, 38
  - Reduce\_Scc, 38
  - Reduce\_transition\_Del\_Sim, 38
  - Reduce\_transition\_Dir\_Sim, 38
- tgba\_reduction
  - dn\_v, 38
  - free\_relation\_simulation, 38
  - get\_delayed\_relation\_simulation, 39
  - get\_direct\_relation\_simulation, 39
  - reduc\_tgba\_sim, 39
  - reduce\_tgba\_options, 38
  - s\_v, 38
  - sn\_v, 38
- tgba\_run
  - print\_tgba\_run, 48
  - project\_tgba\_run, 49
  - reduce\_run, 49
  - replay\_tgba\_run, 49
  - spot::tgba\_run, 362, 363
  - tgba\_run\_to\_tgba, 49
- tgba\_run\_dotty\_decorator
  - spot::tgba\_run\_dotty\_decorator, 365
- tgba\_run\_to\_tgba



- tgba\_run, 49
- tgba\_save\_reachable
  - tgba\_io, 34
- tgba\_sba\_proxy
  - spot::tgba\_sba\_proxy, 370
- tgba\_state\_
  - spot::parity\_game\_graph, 234
  - spot::parity\_game\_graph\_delayed, 241
  - spot::parity\_game\_graph\_direct, 247
- tgba\_succ\_iterator\_concrete
  - spot::tgba\_succ\_iterator\_concrete, 378
- tgba\_succ\_iterator\_product
  - spot::tgba\_succ\_iterator\_product, 383
- tgba\_tba\_proxy
  - spot::tgba\_tba\_proxy, 388
- tgba\_tba\_proxy::operator=
  - spot::tgba\_tba\_proxy, 390
- tgba\_to\_evtgba
  - spot, 70
  - tgba2evtgba.hh, 423
- tgbaalgos/ Directory Reference, 56
- tgbaalgos/bfssteps.hh, 473
- tgbaalgos/dotty.hh, 420
- tgbaalgos/dottydec.hh, 474
- tgbaalgos/dupeexp.hh, 474
- tgbaalgos/emptiness.hh, 475
- tgbaalgos/emptiness\_stats.hh, 476
- tgbaalgos/gtec/ Directory Reference, 52
- tgbaalgos/gtec/ce.hh, 476
- tgbaalgos/gtec/explsc.hh, 477
- tgbaalgos/gtec/gtec.hh, 478
- tgbaalgos/gtec/nsheap.hh, 479
- tgbaalgos/gtec/scstack.hh, 480
- tgbaalgos/gtec/status.hh, 481
- tgbaalgos/gv04.hh, 482
- tgbaalgos/lbtt.hh, 482
- tgbaalgos/ltl2tgba\_fm.hh, 482
- tgbaalgos/ltl2tgba\_lacim.hh, 483
- tgbaalgos/magic.hh, 484
- tgbaalgos/neverclaim.hh, 484
- tgbaalgos/powerset.hh, 485
- tgbaalgos/projrun.hh, 486
- tgbaalgos/randomgraph.hh, 487
- tgbaalgos/reachiter.hh, 421
- tgbaalgos/reducerun.hh, 487
- tgbaalgos/reductgba\_sim.hh, 487
- tgbaalgos/replayrun.hh, 489
- tgbaalgos/rundotdec.hh, 489
- tgbaalgos/save.hh, 423
- tgbaalgos/se05.hh, 490
- tgbaalgos/stats.hh, 490
- tgbaalgos/tau03.hh, 491
- tgbaalgos/tau03opt.hh, 491
- tgbaalgos/weight.hh, 491
- tgparse/ Directory Reference, 57
- tgparse/public.hh, 427
- tgbareduc.hh
  - simulation\_relation, 472
  - state\_couple, 472
- the\_acceptance\_cond\_
  - spot::tgba\_tba\_proxy, 391
- ThirdStep
  - spot::minato\_isop::local\_vars, 211
- time\_info
  - spot::time\_info, 391
- timer
  - spot::timer\_map, 395
- tm
  - spot::timer\_map, 395
- tm\_type
  - spot::timer\_map, 394
- to\_spin\_string
  - ltl\_io, 21
- to\_string
  - ltl\_io, 22
  - spot::duplicator\_node, 152
  - spot::duplicator\_node\_delayed, 157
  - spot::spoiler\_node, 270
  - spot::spoiler\_node\_delayed, 273
- todo
  - spot::couvereur99\_check\_shy, 137
  - spot::evtgba\_reachable\_iterator\_breadth\_first, 185
  - spot::evtgba\_reachable\_iterator\_depth\_first, 188
  - spot::parity\_game\_graph, 234
  - spot::parity\_game\_graph\_delayed, 241
  - spot::parity\_game\_graph\_direct, 247
  - spot::tgba\_reachable\_iterator\_breadth\_first, 343
  - spot::tgba\_reachable\_iterator\_depth\_first, 347
  - spot::tgba\_reduc, 362
- todo\_
  - spot::minato\_isop, 210
- todo\_item
  - spot::couvereur99\_check\_shy::todo\_item, 139
- todo\_list
  - spot::couvereur99\_check\_shy, 135
- top
  - spot::scc\_stack, 262, 263
- total\_
  - spot::timer, 393
- total\_1\_
  - spot::ltl::random\_ltl, 254
- total\_2\_
  - spot::ltl::random\_ltl, 254
- total\_2\_and\_more\_
  - spot::ltl::random\_ltl, 254

- transition\_annotation
  - spot::tgba, 298
  - spot::tgba\_bdd\_concrete, 305
  - spot::tgba\_explicit, 323
  - spot::tgba\_product, 334
  - spot::tgba\_reduc, 360
  - spot::tgba\_sba\_proxy, 372
  - spot::tgba\_tba\_proxy, 390
- transition\_list
  - spot::evtgba\_explicit, 170
- transitions
  - spot::couvreur99\_check, 128
  - spot::couvreur99\_check\_shy, 137
  - spot::ec\_statistics, 159
  - spot::tgba\_statistics, 373
- transitions\_
  - spot::ec\_statistics, 160
- Translating LTL formulae into TGBA, 35
- True
  - spot::ltl::constant, 122
- true\_instance
  - spot::ltl::constant, 123
- tunabbrev.hh
  - unabbreviate\_ltl, 449
- type
  - spot::ltl::binop, 109
  - spot::ltl::constant, 122
  - spot::ltl::multop, 217
  - spot::ltl::unop, 406
- U
  - spot::ltl::binop, 110
- unabbreviate\_logic
  - ltl\_rewriting, 24
- unabbreviate\_logic\_visitor
  - spot::ltl::unabbreviate\_logic\_visitor, 398
- unabbreviate\_ltl
  - spot::ltl, 74
  - tunabbrev.hh, 449
- unabbreviate\_ltl\_visitor
  - spot::ltl::unabbreviate\_ltl\_visitor, 401
- unop
  - spot::ltl::unop, 406
- unref
  - spot::ltl::atomic\_prop, 83
  - spot::ltl::binop, 111
  - spot::ltl::constant, 123
  - spot::ltl::formula, 194
  - spot::ltl::multop, 219
  - spot::ltl::ref\_formula, 259
  - spot::ltl::unop, 408
  - spot::symbol, 292
- unref\_
  - spot::ltl::atomic\_prop, 83
- spot::ltl::binop, 111
- spot::ltl::constant, 124
- spot::ltl::formula, 194
- spot::ltl::multop, 219
- spot::ltl::ref\_formula, 259
- spot::ltl::unop, 408
- unregister\_all\_my\_variables
  - spot::bdd\_dict, 98
- unregister\_variable
  - spot::bdd\_dict, 98
- update\_sums
  - spot::ltl::random\_ltl, 254
- utime
  - spot::time\_info, 391
  - spot::timer, 393
- v1
  - spot::minato\_isop::local\_vars, 212
- val
  - spot::ltl::constant, 124
- val\_
  - spot::ltl::constant, 124
- val\_name
  - spot::ltl::constant, 124
- var\_formula\_map
  - spot::bdd\_dict, 99
- var\_map
  - spot::bdd\_dict, 99
- var\_refs
  - spot::bdd\_dict, 99
- var\_set
  - spot::tgba\_bdd\_core\_data, 314
- varandnext\_set
  - spot::tgba\_bdd\_core\_data, 314
- vars
  - spot::minato\_isop::local\_vars, 212
- vec
  - spot::ltl::multop, 216
- version
  - misc\_tools, 28
- vf\_map
  - spot::bdd\_dict, 95
- visit
  - spot::ltl::clone\_visitor, 114, 115
  - spot::ltl::const\_visitor, 120
  - spot::ltl::postfix\_visitor, 250, 251
  - spot::ltl::simplify\_f\_g\_visitor, 266
  - spot::ltl::unabbreviate\_logic\_visitor, 398
  - spot::ltl::unabbreviate\_ltl\_visitor, 401, 402
  - spot::ltl::visitor, 410
- vr\_map
  - spot::bdd\_dict, 95
- wang32\_hash

- hash\_funcs, 29
- weight
  - spot::weight, 411
- weight\_vector
  - spot::weight, 411
- where\_
  - spot::gspn\_exeption, 200
- X
  - spot::ltd::unop, 406
- Xor
  - spot::ltd::binop, 109
- yy, 74
  - operator+, 75
  - operator+=", 75
  - operator-, 75
  - operator-=, 75
  - operator<<, 75
- yy::Location, 204
  - begin, 206
  - columns, 205
  - end, 206
  - lines, 205
  - Location, 205
  - step, 205
- yy::Position, 247
  - column, 248
  - columns, 248
  - filename, 248
  - initial\_column, 248
  - initial\_line, 248
  - line, 248
  - lines, 248
  - Position, 248
- yy::Slice, 267
  - operator[], 267
  - range\_, 267
  - Slice, 267
  - stack\_, 267
- yy::Stack, 274
  - begin, 276
  - ConstIterator, 275
  - end, 276
  - height, 276
  - Iterator, 275
  - operator[], 276
  - pop, 276
  - push, 276
  - seq\_, 276
  - Stack, 276