

# spot Reference Manual

0.0v

Generated by Doxygen 1.3.7

Tue Jun 29 17:00:35 2004

## Contents

<a href="#">1 spot Main Page</a>	1
<a href="#">2 spot Namespace Index</a>	1
<a href="#">3 spot Hierarchical Index</a>	2
<a href="#">4 spot Class Index</a>	5
<a href="#">5 spot File Index</a>	8
<a href="#">6 spot Namespace Documentation</a>	11
<a href="#">7 spot Class Documentation</a>	34
<a href="#">8 spot File Documentation</a>	283

## 1 spot Main Page

This main page has yet to be written.

### 1.1 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::ltl\\_to\\_tgba\\_lacim](#) Likewise.

## 2 spot Namespace Index

### 2.1 spot Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">spot</a>	11
<a href="#">spot::ltl</a>	25
<a href="#">yy</a>	32

## 3 spot Hierarchical Index

### 3.1 spot Class Hierarchy

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

<code>spot::bdd_less_than</code>	56
<code>spot::ltl::const_visitor</code>	69
<code>spot::counter_example</code>	74
<code>spot::emptiness_check</code>	89
<code>spot::emptiness_check_shy</code>	92
<code>spot::emptiness_check_shy::successor</code>	95
<code>spot::emptiness_check_status</code>	96
<code>spot::ltl::environment</code>	98
<code>spot::ltl::declarative_environment</code>	76
<code>spot::ltl::default_environment</code>	78
<code>spot::explicit_connected_component_factory</code>	101
<code>spot::connected_component_hash_set_factory</code>	67
<code>spot::ltl::formula</code>	102
<code>spot::ltl::constant</code>	70
<code>spot::ltl::ref_formula</code>	159
<code>spot::ltl::atomic_prop</code>	34
<code>spot::ltl::binop</code>	57
<code>spot::ltl::multop</code>	121
<code>spot::ltl::unop</code>	277
<code>spot::free_list</code>	105
<code>spot::bdd_allocator</code>	39
<code>spot::bdd_dict</code>	44
<code>spot::bdd_dict::anon_free_list</code>	53
<code>spot::gspn_exeption</code>	108
<code>spot::gspn_interface</code>	109
<code>spot::gspn_ssp_interface</code>	110

yy::Location	112
spot::magic_search	114
spot::magic_search::magic	117
spot::magic_search::magic_state	117
spot::minato_isop	118
spot::minato_isop::local_vars	120
spot::ltl::multop::paircmp	127
spot::numbered_state_heap	128
spot::numbered_state_heap_hash_map	132
spot::numbered_state_heap_const_iterator	130
spot::numbered_state_heap_factory	131
spot::numbered_state_heap_hash_map_factory	136
yy::Position	155
spot::ptr_hash< T >	159
spot::scc_stack	162
spot::scc_stack::connected_component	164
spot::explicit_connected_component	99
spot::connected_component_hash_set	65
yy::Slice< T, S >	168
spot::spoiler_node	169
spot::duplicator_node	80
spot::duplicator_node_delayed	85
spot::spoiler_node_delayed	171
yy::Stack< T, S >	175
spot::state	177
spot::state_bdd	179
spot::state_explicit	181
spot::state_product	184
spot::state_ptr_equal	187

spot::state_ptr_hash	187
spot::state_ptr_less_than	188
spot::string_hash	188
spot::tgba	189
spot::tgba_bdd_concrete	194
spot::tgba_explicit	209
spot::tgba_reduc	240
spot::tgba_product	221
spot::tgba_tba_proxy	265
spot::tgba_bdd_core_data	203
spot::tgba_bdd_factory	208
spot::tgba_bdd_concrete_factory	200
spot::tgba_explicit::transition	217
spot::tgba_reachable_iterator	228
spot::tgba_reachable_iterator_breadth_first	232
spot::parity_game_graph	137
spot::parity_game_graph_delayed	143
spot::parity_game_graph_direct	149
spot::tgba_reduc	240
spot::tgba_reachable_iterator_depth_first	236
spot::tgba_statistics	254
spot::tgba_succ_iterator	254
spot::tgba_explicit_succ_iterator	218
spot::tgba_succ_iterator_concrete	257
spot::tgba_succ_iterator_product	261
spot::ltl::visitor	282
spot::ltl::clone_visitor	62
spot::ltl::simplify_f_g_visitor	165
spot::ltl::unabbreviate_logic_visitor	271

<code>spot::ltl::unabbreviate_ltl_visitor</code>	274
<code>spot::ltl::postfix_visitor</code>	156

## 4 spot Class Index

### 4.1 spot Class List

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

<code>spot::ltl::atomic_prop</code> (Atomic propositions )	34
<code>spot::bdd_allocator</code> (Manage ranges of variables )	39
<code>spot::bdd_dict</code> (Map BDD variables to formulae )	44
<code>spot::bdd_dict::annon_free_list</code>	53
<code>spot::bdd_less_than</code> (Comparison functor for BDDs )	56
<code>spot::ltl::binop</code> (Binary operator )	57
<code>spot::ltl::clone_visitor</code> (Clone a formula )	62
<code>spot::connected_component_hash_set</code>	65
<code>spot::connected_component_hash_set_factory</code> (Factory for <code>connected_component_hash_set</code> )	67
<code>spot::ltl::const_visitor</code> (Formula visitor that cannot modify the formula )	69
<code>spot::ltl::constant</code> (A constant (True or False) )	70
<code>spot::counter_example</code> (Compute a counter example from a <code>spot::emptiness_check_status</code> )	74
<code>spot::ltl::declarative_environment</code> (A declarative environment )	76
<code>spot::ltl::default_environment</code> (A laxist environment )	78
<code>spot::duplicator_node</code> (Duplicator node of parity game graph )	80
<code>spot::duplicator_node_delayed</code> (Duplicator node of parity game graph for delayed simulation )	85
<code>spot::emptiness_check</code> (Check whether the language of an automate is empty )	89
<code>spot::emptiness_check_shy</code> (A version of <code>spot::emptiness_check</code> try to visit known states first )	92
<code>spot::emptiness_check_shy::successor</code>	95
<code>spot::emptiness_check_status</code> (The status of the emptiness-check on success )	96
<code>spot::ltl::environment</code> (An environment that describes atomic propositions )	98
<code>spot::explicit_connected_component</code> (An SCC storing all its states explicitly )	99

<a href="#">spot::explicit_connected_component_factory</a> (Abstract factory for <a href="#">explicit_connected_component</a> )	101
<a href="#">spot::ltl::formula</a> (An LTL formula )	102
<a href="#">spot::free_list</a> (Manage list of free integers )	105
<a href="#">spot::gspn_exeption</a> (An exeption used to forward GSPN errors )	108
<a href="#">spot::gspn_interface</a>	109
<a href="#">spot::gspn_ssp_interface</a>	110
<a href="#">yy::Location</a> (Abstract a <a href="#">Location</a> )	112
<a href="#">spot::magic_search</a> (Emptiness check on <a href="#">spot::tgba_tba_proxy</a> automata using the Magic Search algorithm )	114
<a href="#">spot::magic_search::magic</a> (Records whether a state has be seen with the magic bit on or off )	117
<a href="#">spot::magic_search::magic_state</a> (A state for the <a href="#">spot::magic_search</a> algorithm )	117
<a href="#">spot::minato_isop</a> (Generate an irredundant sum-of-products (ISOP) form of a BDD function )	118
<a href="#">spot::minato_isop::local_vars</a> (Internal variables for <a href="#">minato_isop</a> )	120
<a href="#">spot::ltl::multop</a> (Multi-operand operators )	121
<a href="#">spot::ltl::multop::paircmp</a> (Comparison functor used internally by <a href="#">ltl::multop</a> )	127
<a href="#">spot::numbered_state_heap</a> (Keep track of a large quantity of indexed states )	128
<a href="#">spot::numbered_state_heap_const_iterator</a> (Iterator on <a href="#">numbered_state_heap</a> objects )	130
<a href="#">spot::numbered_state_heap_factory</a> (Abstract factory for <a href="#">numbered_state_heap</a> )	131
<a href="#">spot::numbered_state_heap_hash_map</a> (A straightforward implementation of <a href="#">numbered_state_heap</a> with a hash map )	132
<a href="#">spot::numbered_state_heap_hash_map_factory</a> (Factory for <a href="#">numbered_state_heap_hash_map</a> )	136
<a href="#">spot::parity_game_graph</a> (Parity game graph which compute a simulation relation )	137
<a href="#">spot::parity_game_graph_delayed</a>	143
<a href="#">spot::parity_game_graph_direct</a> (Parity game graph which compute the direct simulation relation )	149
<a href="#">yy::Position</a> (Abstract a <a href="#">Position</a> )	155
<a href="#">spot::ltl::postfix_visitor</a> (Apply an algorithm on each node of an AST, during a postfix traversal )	156
<a href="#">spot::ptr_hash&lt; T &gt;</a> (A hash function for pointers )	159

<a href="#">spot::ltl::ref_formula</a> (A reference-counted LTL formula )	159
<a href="#">spot::scc_stack</a>	162
<a href="#">spot::scc_stack::connected_component</a>	164
<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> )	165
<a href="#">yy::Slice&lt; T, S &gt;</a>	168
<a href="#">spot::spoiler_node</a> (Spoiler node of parity game graph )	169
<a href="#">spot::spoiler_node_delayed</a> (Spoiler node of parity game graph for delayed simulation )	171
<a href="#">yy::Stack&lt; T, S &gt;</a>	175
<a href="#">spot::state</a> (Abstract class for states )	177
<a href="#">spot::state_bdd</a> (A state whose representation is a BDD )	179
<a href="#">spot::state_explicit</a> (States used by <a href="#">spot::tgba_explicit</a> )	181
<a href="#">spot::state_product</a> (A state for <a href="#">spot::tgba_product</a> )	184
<a href="#">spot::state_ptr_equal</a> (An Equivalence Relation for <code>state*</code> )	187
<a href="#">spot::state_ptr_hash</a> (Hash Function for <code>state*</code> )	187
<a href="#">spot::state_ptr_less_than</a> (Strict Weak Ordering for <code>state*</code> )	188
<a href="#">spot::string_hash</a> (A hash function for strings )	188
<a href="#">spot::tgba</a> (A Transition-based Generalized Büchi Automaton )	189
<a href="#">spot::tgba_bdd_concrete</a> (A concrete <a href="#">spot::tgba</a> implemented using BDDs )	194
<a href="#">spot::tgba_bdd_concrete_factory</a> (Helper class to build a <a href="#">spot::tgba_bdd_concrete</a> object )	200
<a href="#">spot::tgba_bdd_core_data</a> (Core data for a TGBA encoded using BDDs )	203
<a href="#">spot::tgba_bdd_factory</a> (Abstract class for <a href="#">spot::tgba_bdd_concrete</a> factories )	208
<a href="#">spot::tgba_explicit</a> (Explicit representation of a <a href="#">spot::tgba</a> )	209
<a href="#">spot::tgba_explicit::transition</a> (Explicit transitions (used by <a href="#">spot::tgba_explicit</a> ) )	217
<a href="#">spot::tgba_explicit_succ_iterator</a> (Successor iterators used by <a href="#">spot::tgba_explicit</a> )	218
<a href="#">spot::tgba_product</a> (A lazy product. (States are computed on the fly.) )	221
<a href="#">spot::tgba_reachable_iterator</a> (Iterate over all reachable states of a <a href="#">spot::tgba</a> )	228
<a href="#">spot::tgba_reachable_iterator_breadth_first</a> (An implementation of <a href="#">spot::tgba_reachable_iterator</a> that browses states breadth first )	232
<a href="#">spot::tgba_reachable_iterator_depth_first</a> (An implementation of <a href="#">spot::tgba_reachable_iterator</a> that browses states depth first )	236



<a href="#">spot::tgba_reduc</a>	240
<a href="#">spot::tgba_statistics</a>	254
<a href="#">spot::tgba_succ_iterator</a> (Iterate over the successors of a state )	254
<a href="#">spot::tgba_succ_iterator_concrete</a> (A concrete iterator over successors of a TGBA state )	257
<a href="#">spot::tgba_succ_iterator_product</a> (Iterate over the successors of a product computed on the fly )	261
<a href="#">spot::tgba_tba_proxy</a> (Degeneralize a <a href="#">spot::tgba</a> on the fly )	265
<a href="#">spot::ltl::unabbreviate_logic_visitor</a> (Clone and rewrite a formula to remove most of the abbreviated logical operators )	271
<a href="#">spot::ltl::unabbreviate_ltl_visitor</a> (Clone and rewrite a formula to remove most of the abbreviated LTL and logical operators )	274
<a href="#">spot::ltl::unop</a> (Unary operator )	277
<a href="#">spot::ltl::visitor</a> (Formula visitor that can modify the formula )	282

## 5 spot File Index

### 5.1 spot File List

Here is a list of all files with brief descriptions:

<a href="#">/home/adl/proj/spot/doc/mainpage.dox</a>	283
<a href="#">gspn/common.hh</a>	283
<a href="#">gspn/gspn.hh</a>	284
<a href="#">gspn/ssp.hh</a>	284
<a href="#">ltlast/allnodes.hh</a>	285
<a href="#">ltlast/atomic_prop.hh</a>	285
<a href="#">ltlast/binop.hh</a>	286
<a href="#">ltlast/constant.hh</a>	287
<a href="#">ltlast/formula.hh</a>	288
<a href="#">ltlast/multop.hh</a>	288
<a href="#">ltlast/predecl.hh</a>	289
<a href="#">ltlast/refformula.hh</a>	289
<a href="#">ltlast/unop.hh</a>	290
<a href="#">ltlast/visitor.hh</a>	291

<a href="#">ltlenv/declenv.hh</a>	291
<a href="#">ltlenv/defaultenv.hh</a>	292
<a href="#">ltlenv/environment.hh</a>	293
<a href="#">ltlparse/location.hh</a>	293
<a href="#">ltlparse/position.hh</a>	294
<a href="#">ltlparse/public.hh</a>	295
<a href="#">ltlparse/stack.hh</a>	297
<a href="#">ltlvisit/basicreduce.hh</a>	297
<a href="#">ltlvisit/clone.hh</a>	298
<a href="#">ltlvisit/destroy.hh</a>	299
<a href="#">ltlvisit/dotty.hh</a>	299
<a href="#">ltlvisit/dump.hh</a>	300
<a href="#">ltlvisit/length.hh</a>	301
<a href="#">ltlvisit/lunabbrev.hh</a>	301
<a href="#">ltlvisit/nenoform.hh</a>	302
<a href="#">ltlvisit/postfix.hh</a>	302
<a href="#">ltlvisit/reduce.hh</a>	303
<a href="#">ltlvisit/simpfg.hh</a>	303
<a href="#">ltlvisit/syntimpl.hh</a>	304
<a href="#">ltlvisit/tostring.hh</a>	304
<a href="#">ltlvisit/tunabbrev.hh</a>	305
<a href="#">misc/bddalloc.hh</a>	305
<a href="#">misc/bddlt.hh</a>	306
<a href="#">misc/escape.hh</a>	307
<a href="#">misc/freelist.hh</a>	307
<a href="#">misc/hash.hh</a>	308
<a href="#">misc/minato.hh</a>	308
<a href="#">misc/version.hh</a>	309
<a href="#">tgba/bdddict.hh</a>	309

<a href="#">tgba/bddprint.hh</a>	310
<a href="#">tgba/formula2bdd.hh</a>	310
<a href="#">tgba/public.hh</a>	296
<a href="#">tgba/state.hh</a>	311
<a href="#">tgba/statebdd.hh</a>	312
<a href="#">tgba/succiter.hh</a>	312
<a href="#">tgba/succiterconcrete.hh</a>	313
<a href="#">tgba/tgba.hh</a>	314
<a href="#">tgba/tgababddconcrete.hh</a>	314
<a href="#">tgba/tgababddconcretefactory.hh</a>	315
<a href="#">tgba/tgababddconcreteproduct.hh</a>	316
<a href="#">tgba/tgababddcoredata.hh</a>	317
<a href="#">tgba/tgababddfactory.hh</a>	318
<a href="#">tgba/tgbaexplicit.hh</a>	318
<a href="#">tgba/tgabproduct.hh</a>	319
<a href="#">tgba/tgbareduc.hh</a>	320
<a href="#">tgba/tgbatba.hh</a>	320
<a href="#">tgbaalgos/dotty.hh</a>	300
<a href="#">tgbaalgos/duexp.hh</a>	321
<a href="#">tgbaalgos/lbtt.hh</a>	326
<a href="#">tgbaalgos/ltl2tgba_fm.hh</a>	327
<a href="#">tgbaalgos/ltl2tgba_lacim.hh</a>	328
<a href="#">tgbaalgos/magic.hh</a>	328
<a href="#">tgbaalgos/neverclaim.hh</a>	329
<a href="#">tgbaalgos/powerset.hh</a>	330
<a href="#">tgbaalgos/reachiter.hh</a>	330
<a href="#">tgbaalgos/reductgba_sim.hh</a>	331
<a href="#">tgbaalgos/save.hh</a>	332
<a href="#">tgbaalgos/stats.hh</a>	332

<a href="#">tgbaalgos/gtec/ce.hh</a>	322
<a href="#">tgbaalgos/gtec/explscce.hh</a>	323
<a href="#">tgbaalgos/gtec/gtec.hh</a>	323
<a href="#">tgbaalgos/gtec/nsheap.hh</a>	324
<a href="#">tgbaalgos/gtec/sccestack.hh</a>	325
<a href="#">tgbaalgos/gtec/status.hh</a>	326
<a href="#">tgbaparse/public.hh</a>	296

## 6 spot Namespace Documentation

### 6.1 spot Namespace Reference

#### 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.*
- struct [spot::ptr\\_hash< T >](#)  
*A hash function for pointers.*
- struct [spot::string\\_hash](#)  
*A hash function for strings.*
- class [spot::minato\\_isop](#)  
*Generate an irredundant sum-of-products (ISOP) form of a BDD function.*
- struct [spot::minato\\_isop::local\\_vars](#)  
*Internal variables for [minato\\_isop](#).*
- class [spot::bdd\\_dict](#)  
*Map BDD variables to formulae.*
- class [spot::bdd\\_dict::annon\\_free\\_list](#)
- 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::state\\_bdd](#)  
*A state whose representation is a BDD.*
- class [spot::tgba\\_succ\\_iterator](#)  
*Iterate over the successors of a state.*
- class [spot::tgba\\_succ\\_iterator\\_concrete](#)  
*A concrete iterator over successors of a TGBA state.*
- class [spot::tgba](#)  
*A Transition-based Generalized Büchi Automaton.*
- class [spot::tgba\\_bdd\\_concrete](#)  
*A concrete [spot::tgba](#) implemented using BDDs.*
- class [spot::tgba\\_bdd\\_concrete\\_factory](#)  
*Helper class to build a [spot::tgba\\_bdd\\_concrete](#) object.*
- struct [spot::tgba\\_bdd\\_core\\_data](#)  
*Core data for a TGBA encoded using BDDs.*
- class [spot::tgba\\_bdd\\_factory](#)  
*Abstract class for [spot::tgba\\_bdd\\_concrete](#) factories.*
- class [spot::tgba\\_explicit](#)  
*Explicit representation of a [spot::tgba](#).*
- struct [spot::tgba\\_explicit::transition](#)  
*Explicit transitions (used by [spot::tgba\\_explicit](#)).*
- class [spot::state\\_explicit](#)  
*States used by [spot::tgba\\_explicit](#).*
- class [spot::tgba\\_explicit\\_succ\\_iterator](#)  
*Successor iterators used by [spot::tgba\\_explicit](#).*
- class [spot::state\\_product](#)  
*A state for [spot::tgba\\_product](#).*
- class [spot::tgba\\_succ\\_iterator\\_product](#)  
*Iterate over the successors of a product computed on the fly.*
- class [spot::tgba\\_product](#)  
*A lazy product. (States are computed on the fly.).*

- class `spot::tgba_reduc`
- class `spot::tgba_tba_proxy`  
*Degeneralize a `spot::tgba` on the fly.*
- class `spot::counter_example`  
*Compute a counter example from a `spot::emptiness_check_status`.*
- class `spot::explicit_connected_component`  
*An SCC storing all its states explicitly.*
- class `spot::connected_component_hash_set`
- class `spot::explicit_connected_component_factory`  
*Abstract factory for `explicit_connected_component`.*
- class `spot::connected_component_hash_set_factory`  
*Factory for `connected_component_hash_set`.*
- class `spot::emptiness_check`  
*Check whether the language of an automata is empty.*
- class `spot::emptiness_check_shy`  
*A version of `spot::emptiness_check` try to visit known states first.*
- struct `spot::emptiness_check_shy::successor`
- class `spot::numbered_state_heap_const_iterator`  
*Iterator on `numbered_state_heap` objects.*
- class `spot::numbered_state_heap`  
*Keep track of a large quantity of indexed states.*
- class `spot::numbered_state_heap_factory`  
*Abstract factory for `numbered_state_heap`.*
- class `spot::numbered_state_heap_hash_map`  
*A straightforward implementation of `numbered_state_heap` with a hash map.*
- class `spot::numbered_state_heap_hash_map_factory`  
*Factory for `numbered_state_heap_hash_map`.*
- class `spot::scc_stack`
- struct `spot::scc_stack::connected_component`
- class `spot::emptiness_check_status`  
*The status of the emptiness-check on success.*
- struct `spot::magic_search`  
*Emptiness check on `spot::tgba_tba_proxy` automata using the Magic Search algorithm.*
- struct `spot::magic_search::magic`  
*Records whether a state has be seen with the magic bit on or off.*

- struct `spot::magic_search::magic_state`  
*A state for the `spot::magic_search` algorithm.*
- 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.*
- 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`
- struct `spot::tgba_statistics`
- class `spot::gspn_exeption`  
*An exeption used to forward GSPN errors.*
- class `spot::gspn_interface`
- class `spot::gspn_ssp_interface`

## Typedefs

- 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_Dir_Sim` = 1, `Reduce_Del_Sim` = 2, `Reduce_Scc` = 4,  
    `Reduce_All` = -1U }  
    *Options for reduce.*

## 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.*
- `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 tgba::tgba\_bdd\_concrete automata.*
- `std::ostream & dotted_reachable` (`std::ostream &os`, `const tgba *g`)  
*Print reachable states in dot format.*
- `tgba_explicit * tgba_dupexp_bfs` (`const tgba *aut`)
- `tgba_explicit * tgba_dupexp_dfs` (`const tgba *aut`)
- `std::ostream & lbtt_reachable` (`std::ostream &os`, `const tgba *g`)  
*Print reachable states in LBTT format.*
- `std::ostream & nonacceptant_lbtt_reachable` (`std::ostream &os`, `const tgba *g`)  
*Print an LBTT automaton for statistics.*
- `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`)  
*Build a spot::tgba\_explicit\* from an LTL formula.*
- `tgba_bdd_concrete * ltl_to_tgba_lacim` (`const ltl::formula *f`, `bdd_dict *dict`)
- `std::ostream & never_claim_reachable` (`std::ostream &os`, `const tgba_tba_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 * reduc_tgba_sim` (`const tgba *a`, `int opt=Reduce_All`)  
*the reduced automata.*
- `simulation_relation * get_direct_relation_simulation` (`const tgba *a`, `int opt=-1`)  
*Compute a direct simulation relation on state of tgba f.*
- `simulation_relation * get_delayed_relation_simulation` (`const tgba *a`, `int opt=-1`)
- `void free_relation_simulation` (`simulation_relation *rel`)  
*To free a simulation relation.*
- `bool is_include` (`const tgba *a1`, `const tgba *a2`)
- `std::ostream & tgba_save_reachable` (`std::ostream &os`, `const tgba *g`)  
*Save reachable states in text format.*
- `tgba_statistics stats_reachable` (`const tgba *g`)  
*Compute statistics for an automata.*
- `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, `tgba_parse_error_list` &error\_list)

Format diagnostics produced by `spot::tgba_parse`.

- std::ostream & `operator<<` (std::ostream &os, const `gspn_exeption` &e)
- `emptiness_check` \* `emptiness_check_ssp_semi` (const `tgba` \*ssp\_automata)
- `emptiness_check` \* `emptiness_check_ssp_shy_semi` (const `tgba` \*ssp\_automata)
- `emptiness_check` \* `emptiness_check_ssp_shy` (const `tgba` \*ssp\_automata)
- `counter_example` \* `counter_example_ssp` (const `emptiness_check_status` \*status)

### 6.1.1 Typedef Documentation

**6.1.1.1** typedef Sgi::vector<`duplicator_node`\*> `spot::dn_v`

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

**6.1.1.3** typedef Sgi::vector<`state_couple`\*> `spot::simulation_relation`

**6.1.1.4** typedef Sgi::vector<`spoiler_node`\*> `spot::sn_v`

**6.1.1.5** typedef Sgi::pair<const `spot::state`\*, const `spot::state`\*> `spot::state_couple`

**6.1.1.6** typedef std::pair<`yy::Location`, std::string> `spot::tgba_parse_error`

A parse diagnostic with its location.

**6.1.1.7** typedef std::list<`tgba_parse_error`> `spot::tgba_parse_error_list`

A list of parser diagnostics, as filled by parse.

### 6.1.2 Enumeration Type Documentation

**6.1.2.1** enum `spot::reduce_tgba_options`

Options for reduce.

**Enumeration values:**

*Reduce\_None* No reduction.

*Reduce\_Dir\_Sim* Reduction using direct simulation relation.

*Reduce\_Del\_Sim* Reduction using delayed simulation relation.

*Reduce\_Scc* Reduction using SCC.

*Reduce\_All* All reductions.

### 6.1.3 Function Documentation

#### 6.1.3.1 `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.

#### 6.1.3.2 `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.

#### 6.1.3.3 `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.

#### 6.1.3.4 `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.

**6.1.3.5 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.

**6.1.3.6 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.

**6.1.3.7 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.

**6.1.3.8 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.

**6.1.3.9** `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.

**6.1.3.10** `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.

**6.1.3.11** `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.

**6.1.3.12** `const ltl::formula* bdd_to_formula (bdd f, const bdd_dict * d)`**6.1.3.13** `counter\_example* counter_example_ssp (const emptiness_check_status * status)`**6.1.3.14** `std::ostream& dotted_reachable (std::ostream & os, const tgba * g)`

Print reachable states in dot format.

**6.1.3.15** `emptiness\_check* emptiness_check_ssp_semi (const tgba * ssp_automata)`**6.1.3.16** `emptiness\_check* emptiness_check_ssp_shy (const tgba * ssp_automata)`**6.1.3.17** `emptiness\_check* emptiness_check_ssp_shy_semi (const tgba * ssp_automata)`**6.1.3.18** `std::string escape_str (const std::string & str)`

Escape " and \ characters in *str*.

**6.1.3.19** `std::ostream& escape_str (std::ostream & os, const std::string & str)`

Escape " and \ characters in *str*.

**6.1.3.20** `bool format_tgba_parse_errors (std::ostream & os, tgba_parse_error_list & error_list)`

Format diagnostics produced by [spot::tgba\\_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.

**6.1.3.21** `bdd formula_to_bdd (const ltl::formula * f, bdd_dict * d, void * for_me)`**6.1.3.22** `void free_relation_simulation (simulation_relation * rel)`

To free a simulation relation.

**6.1.3.23** `simulation_relation* get_delayed_relation_simulation (const tgba * a, int opt = -1)`

Compute a delayed simulation relation on state of *tgba f*. **FIXME** : this method is incorrect !! Don't use it !!

**6.1.3.24** `simulation_relation* get_direct_relation_simulation (const tgba * a, int opt = -1)`

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

**6.1.3.25** `bool is_include (const tgba * a1, const tgba * a2)`

Test if the initial state of *a2* fair simulate this of *a1*. Not implemented.

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

Print reachable states in LBTT format.

Note that LBTT expects an automaton with transition labeled by propositional formulae, and generalized Büchi acceptance conditions on **states**. This is unlike our [spot::tgba](#) automata which put both generalized acceptance conditions (and propositional formulae) on **transitions**.

This algorithm will therefore produce an automata where acceptance conditions have been moved from each transition to the previous state. In the worst case, doing so will multiply the number of states and transitions of the automata by  $2^{|Acc|}$ . where  $|Acc|$  is the number of acceptance conditions used by the automata. (It can be a bit more because LBTT allows only for one initial state: [lbtt\\_reachable\(\)](#) may also have to create an additional state in case the source initial state had to be split.) You have been warned.

**Parameters:**

*g* The automata to print.

*os* Where to print.

**6.1.3.27 `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`)

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}
}
```

If `exprop` is 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).

If `symb_merge` is set to false, states with the same symbolic representation (these are equivalent formulae) will not be merged.

If `branching_postponement` is 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}
}
```

If `fair_loop_approx` is set, a really simple characterization of unstable state is used to suppress all acceptance conditions from incoming transitions.

**6.1.3.28 `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}
}

```

### 6.1.3.29 `std::ostream& never_claim_reachable (std::ostream & os, const tgba_tba_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.

### 6.1.3.30 `std::ostream& nonacceptant_lbtt_reachable (std::ostream & os, const tgba * g)`

Print an LBTT automaton for statistics.

Output *g* in LBTT's format but ignoring the acceptance conditions, of all its transitions. This produces an automaton that has the same size as *g*, and whose synchronized product with another automaton also has the same size. This will also declare as much acceptance conditions has there is in *g* (they will just be never used).

The produced automaton will not recognize any word (unless *g* has no acceptance condition, in which case this function is a no-op).

The produced automaton is useful to obtain accurate statistics from LBTT, without any size blow up of the automata.

### 6.1.3.31 `std::ostream& operator<< (std::ostream & os, const gspn_exception & e)`

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

Multiplies two `tgba::tgba_bdd_concrete` automata.

This function build the resulting product, as another `tgba::tgba_bdd_concrete` automaton.

### 6.1.3.33 `tgba* reduc_tgba_sim (const tgba * a, int opt = Reduce_All)`

the reduced automata.

#### Parameters:

- a* the automata to reduce.
- opt* a conjunction of `spot::reduce_tgba_options` specifying



**6.1.3.34** `tgba_statistics stats_reachable (const tgba * g)`

Compute statistics for an automata.

**6.1.3.35** `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.

**6.1.3.36** `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.

**6.1.3.37** `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 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.

**6.1.3.38** `tgba_explicit* tgba_powerset (const tgba * aut)`

Build a deterministic automaton, ignoring acceptance conditions.

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

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

Save reachable states in text format.

**6.1.3.40** `const char* version ()`

Return Spot's version.

## 6.2 spot::ltl Namespace Reference

### 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.*
- struct [spot::ltl::multop::paircmp](#)  
*Comparison functor used internally by [ltl::multop](#).*
- class [spot::ltl::ref\\_formula](#)  
*A reference-counted LTL formula.*
- class [spot::ltl::unop](#)  
*Unary operator.*
- struct [spot::ltl::visitor](#)  
*Formula visitor that can modify the formula.*
- struct [spot::ltl::const\\_visitor](#)  
*Formula visitor that cannot modify the formula.*
- class [spot::ltl::declarative\\_environment](#)  
*A declarative environment.*
- class [spot::ltl::default\\_environment](#)  
*A laxist environment.*
- class [spot::ltl::environment](#)  
*An environment that describes atomic propositions.*
- 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\\_f\\_g\\_visitor](#)  
*Replace `true U 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.*

### 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`.*

### 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 &ltlt_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 &ltlt_string`, [parse\\_error\\_list](#) &error\_list)  
*Format diagnostics produced by `spot::ltl::parse`.*
- [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)  
*the reduced formula*
- bool **is\_eventual** (const **formula** \*f)  
*Check whether a formula is eventual.*
- bool **is\_universal** (const **formula** \*f)  
*Check whether a formula is universal.*
- **formula** \* **simplify\_f\_g** (const **formula** \*f)  
*Replace true  $\cup$  f and false  $\cap$  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.*

### 6.2.1 Typedef Documentation

#### 6.2.1.1 typedef std::pair<yy::Location, std::string> spot::ltl::parse\_error

A parse diagnostic with its location.

#### 6.2.1.2 typedef std::list<parse\_error> spot::ltl::parse\_error\_list

A list of parser diagnostics, as filled by parse.

### 6.2.2 Enumeration Type Documentation

#### 6.2.2.1 enum spot::ltl::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.

### 6.2.3 Function Documentation

#### 6.2.3.1 [formula\\*](#) basic\_reduce (const formula \*f)

Basic rewritings.

#### 6.2.3.2 [formula\\*](#) clone (const formula \*f)

Clone a formula.

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

Destroys a formula.

#### 6.2.3.4 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/>

**6.2.3.5 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.

**6.2.3.6 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.

**6.2.3.7 bool is\_eventual (const formula \**f*)**

Check whether a formula is eventual.

This comes from

```
@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}
}
```

FIXME: Describe what eventual formulae are.

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

Whether a formula starts with FG.

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

Whether a formula starts with GF.

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

Check whether a formula is universal.

FIXME: Describe what universal formulae are. Cite paper.

**6.2.3.11 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.

**6.2.3.12 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.)

**6.2.3.13 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 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.

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

the reduced formula

**Parameters:**

*f* the formula to reduce

*opt* a conjunction of `spot::ltl::reduce_options` specifying

**6.2.3.15 formula\* simplify\_f\_g (const formula \*f)**

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

**6.2.3.16 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}
}
```

**6.2.3.17 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.

**6.2.3.18 std::string to\_spin\_string (const formula \*f)**

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

**Parameters:**

*f* The formula to translate.

**6.2.3.19 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.



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

Convert a formula into a (parsable) string.

**Parameters:**

*f* The formula to translate.

**6.2.3.21** `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.

**6.2.3.22** `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`.

**6.2.3.23** `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`.

**6.3 yy Namespace Reference****Classes**

- class `yy::Location`  
*Abstract a `Location`.*
- class `yy::Position`  
*Abstract a `Position`.*
- class `yy::Stack< T, S >`
- class `yy::Slice< T, S >`

**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 width)  
*Add two `Location` objects.*

- `Location & operator+= (Location &res, unsigned 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.*

### 6.3.1 Function Documentation

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

Add two [Position](#) objects.

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

Add two [Location](#) objects.

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

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

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

Add and assign a [Position](#).

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

Add and assign a [Location](#).

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

Add two [Position](#) objects.

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

Add and assign a [Position](#).

### 6.3.1.8 `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

### 6.3.1.9 `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.

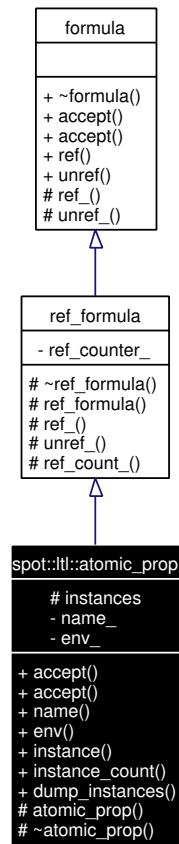
## 7 spot Class Documentation

### 7.1 `spot::ltl::atomic_prop` Class Reference

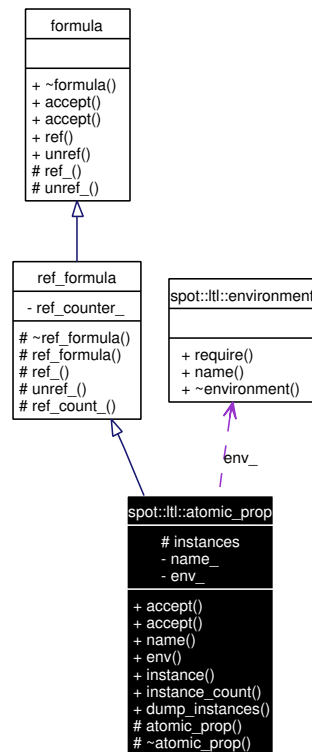
Atomic propositions.

```
#include <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*

## Static Public Member Functions

- `atomic_prop * instance` (const std::string &name, `environment` &env)
- unsigned `instance_count ()`  
*Number of instantiated atomic propositions. For debugging.*
- std::ostream & `dump_instances` (std::ostream &os)

*List all instances of atomic propositions. For debugging.*

- 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.*

### Static Protected Attributes

- [map](#) instances

### Private Attributes

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

#### 7.1.1 Detailed Description

Atomic propositions.

#### 7.1.2 Member Typedef Documentation

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

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

### 7.1.3 Constructor & Destructor Documentation

**7.1.3.1** `spot::ltl::atomic_prop::atomic_prop (const std::string & name, environment & env)` `[protected]`

**7.1.3.2** `virtual spot::ltl::atomic_prop::~~atomic_prop ()` `[protected, virtual]`

### 7.1.4 Member Function Documentation

**7.1.4.1** `virtual void spot::ltl::atomic_prop::accept (const\_visitor & visitor) const` `[virtual]`

Entry point for `vspot::ltl::const_visitor` instances.

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

**7.1.4.2** `virtual void spot::ltl::atomic_prop::accept (visitor & visitor)` `[virtual]`

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

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

**7.1.4.3** `std::ostream& spot::ltl::atomic_prop::dump_instances (std::ostream & os)` `[static]`

List all instances of atomic propositions. For debugging.

**7.1.4.4** `environment& spot::ltl::atomic_prop::env () const`

Get the environment of the atomic proposition.

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

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

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

Number of instantiated atomic propositions. For debugging.

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

Get the name of the atomic proposition.

**7.1.4.8** `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.

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

increment reference counter if any

Reimplemented from [spot::ltl::formula](#).

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

Number of references to this formula.

**7.1.4.11** `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.

**7.1.4.12** `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](#).

**7.1.5 Member Data Documentation****7.1.5.1** `environment*` [spot::ltl::atomic\\_prop::env\\_](#) [private]**7.1.5.2** `map` [spot::ltl::atomic\\_prop::instances](#) [static, protected]**7.1.5.3** `std::string` [spot::ltl::atomic\\_prop::name\\_](#) [private]

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

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

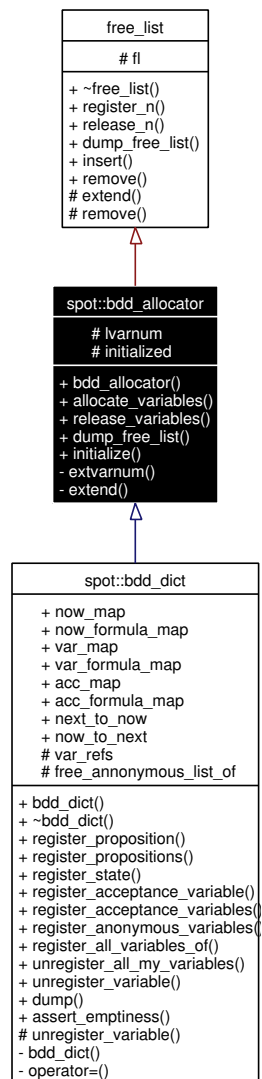
**7.2 `spot::bdd_allocator` Class Reference**

Manage ranges of variables.

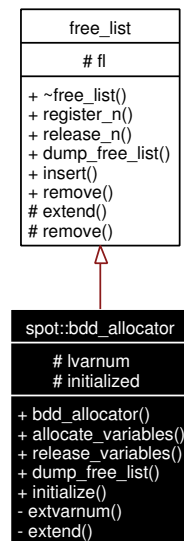
```
#include <bddalloc.hh>
```

Inheritance diagram for `spot::bdd_allocator`:





Collaboration diagram for `spot::bdd_allocator`:



### Public Member Functions

- [bdd\\_allocator](#) ()  
*Default constructor.*
- [int allocate\\_variables](#) (int n)  
*Allocate n BDD variables.*
- [void release\\_variables](#) (int base, int n)  
*Release n BDD variables starting at base.*
- [std::ostream & dump\\_free\\_list](#) (std::ostream &os) const  
*Dump the list to os for debugging.*

### Static Public Member Functions

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

### Protected Attributes

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

### Static Protected Attributes

- [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.*
- 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.*

## 7.2.1 Detailed Description

Manage ranges of variables.

## 7.2.2 Member Typedef Documentation

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

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

Such pairs describe second free integer starting at first.

### 7.2.3 Constructor & Destructor Documentation

**7.2.3.1** `spot::bdd_allocator::bdd_allocator ()`

Default constructor.

### 7.2.4 Member Function Documentation

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

Allocate *n* BDD variables.

**7.2.4.2** `std::ostream& spot::free_list::dump_free_list (std::ostream & os) const`

Dump the list to *os* for debugging.

**7.2.4.3** `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`.

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

Require more variables.

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

Initialize the BDD library.

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

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

**7.2.4.7** `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

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

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

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

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

**7.2.4.10 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*.

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

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

**7.2.5 Member Data Documentation****7.2.5.1 free\_list\_type spot::free\_list::fl** [protected, inherited]

Tracks unused BDD variables.

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

Whether the BDD library has been initialized.

**7.2.5.3 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:

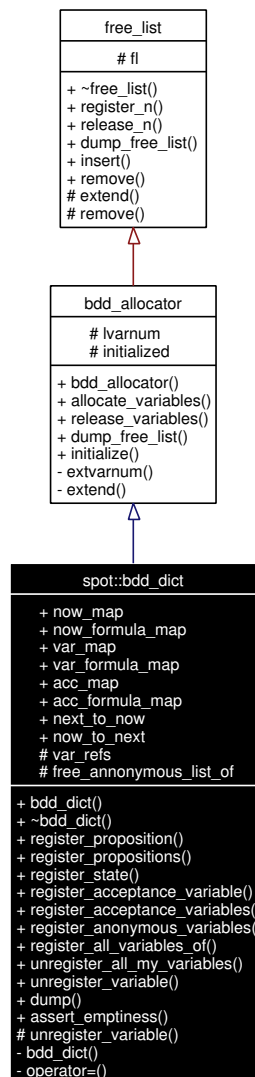
- [misc/bddalloc.hh](#)

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

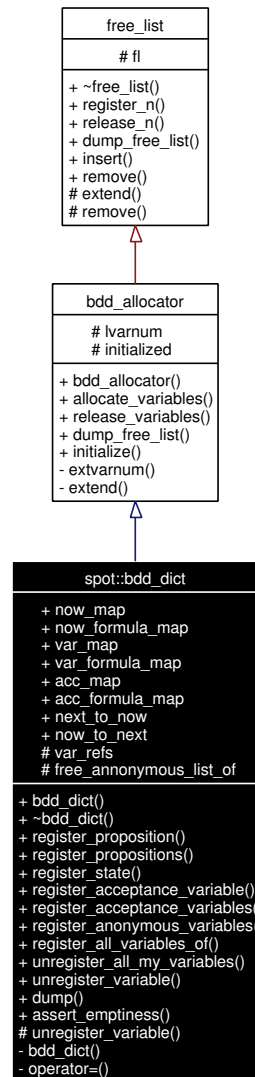
Map BDD variables to formulae.

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

Inheritance diagram for spot::bdd\_dict:



Collaboration diagram for spot::bdd\_dict:



## Public Types

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

## Public Member Functions

- [bdd\\_dict](#) ()
- [~bdd\\_dict](#) ()
- int [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.*
- std::ostream & [dump\\_free\\_list](#) (std::ostream &os) const  
*Dump the list to os for debugging.*
- 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

- 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 Sgi::hash_set< const void *, ptr\_hash< void > > ref\_set`  
*BDD-variable reference counts.*
- `typedef Sgi::hash_map< int, ref\_set > vr\_map`
- `typedef Sgi::hash_map< const void *, annon\_free\_list, ptr\_hash< void > > 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

- 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)

#### 7.3.1 Detailed Description

Map BDD variables to formulae.

#### 7.3.2 Member Typedef Documentation

**7.3.2.1** `typedef Sgi::hash_map<const void*, annon\_free\_list, ptr\_hash<void> > spot::bdd\_dict::free\_anonymous\_list\_of\_type [protected]`

List of unused anonymous variable number for each automaton.

**7.3.2.2** `typedef Sgi::hash_map<const ltl::formula\*, int, ptr\_hash<ltl::formula> > spot::bdd\_dict::fv\_map`

Formula-to-BDD-variable maps.

**7.3.2.3** `typedef Sgi::hash_set<const void*, ptr\_hash<void> > spot::bdd\_dict::ref\_set [protected]`

BDD-variable reference counts.

**7.3.2.4** `typedef Sgi::hash_map<int, const ltl::formula\*> spot::bdd\_dict::vf\_map`

BDD-variable-to-formula maps.

**7.3.2.5** `typedef Sgi::hash_map<int, ref\_set> spot::bdd\_dict::vr\_map [protected]`

#### 7.3.3 Constructor & Destructor Documentation

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

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

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

### 7.3.4 Member Function Documentation

#### 7.3.4.1 `int spot::bdd_allocator::allocate_variables (int n)` `[inherited]`

Allocate *n* BDD variables.

#### 7.3.4.2 `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.

#### 7.3.4.3 `std::ostream& spot::bdd_dict::dump (std::ostream & os) const`

Dump all variables for debugging.

#### Parameters:

*os* The output stream.

#### 7.3.4.4 `std::ostream& spot::free_list::dump_free_list (std::ostream & os) const` `[inherited]`

Dump the list to *os* for debugging.

#### 7.3.4.5 `void spot::bdd_allocator::initialize ()` `[static, inherited]`

Initialize the BDD library.

#### 7.3.4.6 `bool spot::bdd_dict::is_registered_acceptance_variable (const ltl::formula * f, const void * by_me)`

#### 7.3.4.7 `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*.

#### 7.3.4.8 `bool spot::bdd_dict::is_registered_state (const ltl::formula * f, const void * by_me)`

#### 7.3.4.9 `bdd\_dict& spot::bdd_dict::operator= (const bdd\_dict & other)` `[private]`

#### 7.3.4.10 `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.

**7.3.4.11 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).

**7.3.4.12 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.

**7.3.4.13 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.

**7.3.4.14 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.

**7.3.4.15 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).

**7.3.4.16 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.

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

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

**7.3.4.18** `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*.

**7.3.4.19** `void spot::bdd_dict::unregister_variable (vr_map::iterator & cur, const void * me)`  
[protected]**7.3.4.20** `void spot::bdd_dict::unregister_variable (int var, const void * me)`

Release a variable used by *me*.

**7.3.5 Member Data Documentation****7.3.5.1** `vf_map spot::bdd_dict::acc_formula_map`

Maps BDD variables to acceptance conditions.

**7.3.5.2** `fv_map spot::bdd_dict::acc_map`

Maps acceptance conditions to BDD variables.

**7.3.5.3** `free_anonymous_list_of_type spot::bdd_dict::free_anonymous_list_of` [protected]**7.3.5.4** `bool spot::bdd_allocator::initialized` [static, protected, inherited]

Whether the BDD library has been initialized.

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

number of variables in use in this allocator.

**7.3.5.6** `bddPair* spot::bdd_dict::next_to_now`

Map Next variables to Now variables.

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

**7.3.5.7** `vf_map spot::bdd_dict::now_formula_map`

Maps "Now" BDD variables to formulae.

**7.3.5.8** `fv_map spot::bdd_dict::now_map`

Maps formulae to "Now" BDD variables.

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

Map Now variables to Next variables.

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

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

Maps BDD variables to atomic propositions.

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

Maps atomic propositions to BDD variables.

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

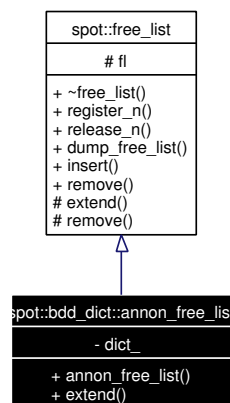
The documentation for this class was generated from the following file:

- [tgba/bdddict.hh](#)

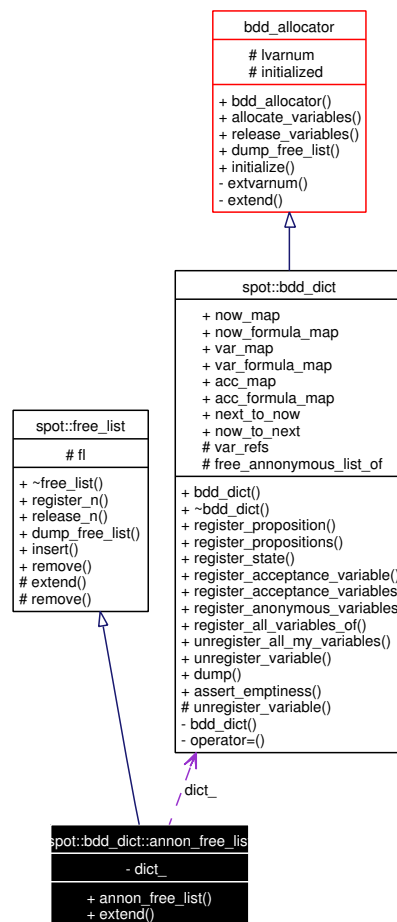
**7.4 spot::bdd\_dict::annon\_free\_list Class Reference**

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

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



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



## Public Member Functions

- `annon_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\_

#### 7.4.1 Member Typedef Documentation

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

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

Such pairs describe second free integer starting at first.

#### 7.4.2 Constructor & Destructor Documentation

**7.4.2.1** spot::bdd\_dict::annon\_free\_list::annon\_free\_list ([bdd\\_dict](#) \* d = 0)

#### 7.4.3 Member Function Documentation

**7.4.3.1** std::ostream& spot::free\_list::dump\_free\_list (std::ostream & os) const [inherited]

Dump the list to os for debugging.

**7.4.3.2** 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](#).

#### 7.4.3.3 void spot::free\_list::insert (int *base*, int *n*) [inherited]

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

#### 7.4.3.4 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

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

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

#### 7.4.3.6 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*.

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

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

### 7.4.4 Member Data Documentation

#### 7.4.4.1 bdd\_dict\* spot::bdd\_dict::annon\_free\_list::dict\_ [private]

#### 7.4.4.2 free\_list\_type spot::free\_list::fl [protected, inherited]

Tracks unused BDD variables.

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

- [tgba/bdddict.hh](#)

## 7.5 spot::bdd\_less\_than Struct Reference

Comparison functor for BDDs.

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

### Public Member Functions

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

### 7.5.1 Detailed Description

Comparison functor for BDDs.

### 7.5.2 Member Function Documentation

**7.5.2.1** `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:

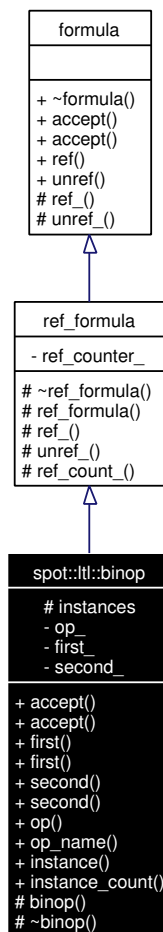
- [misc/bddlt.hh](#)

## 7.6 spot::ltl::binop Class Reference

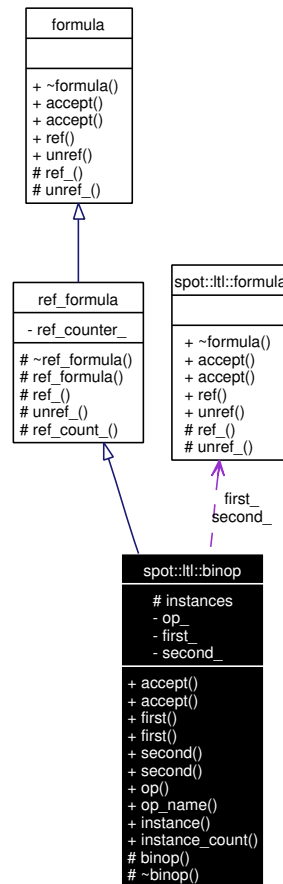
Binary operator.

```
#include <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*

### Static Public Member Functions

- binop \* instance (type op, formula \*first, formula \*second)

- unsigned instance\_count ()

*Number of instantiated binary operators. For debugging.*

- 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.*

### Static Protected Attributes

- [map instances](#)

### Private Attributes

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

## 7.6.1 Detailed Description

Binary operator.

## 7.6.2 Member Typedef Documentation

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

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

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

## 7.6.3 Member Enumeration Documentation

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

Different kinds of binary operators

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

Enumeration values:

*Xor*

*Implies*

*Equiv*

*U*

*R*

## 7.6.4 Constructor & Destructor Documentation

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

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

## 7.6.5 Member Function Documentation

**7.6.5.1** `virtual void spot::ltl::binop::accept (const\_visitor & v) const [virtual]`

Entry point for `vspot::ltl::const_visitor` instances.

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

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

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

Implements `spot::ltl::formula`.

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

Get the first operand.

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

Get the first operand.

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

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

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

Number of instantiated binary operators. For debugging.

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

Get the type of this operator.

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

Get the type of this operator, as a string.

**7.6.5.9** `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.

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

increment reference counter if any

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

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

Number of references to this formula.

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

Get the second operand.

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

Get the second operand.

**7.6.5.14** `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.

**7.6.5.15** `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`.

**7.6.6** Member Data Documentation**7.6.6.1** `formula* spot::ltl::binop::first_` [private]**7.6.6.2** `map spot::ltl::binop::instances` [static, protected]**7.6.6.3** `type spot::ltl::binop::op_` [private]**7.6.6.4** `formula* spot::ltl::binop::second_` [private]

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

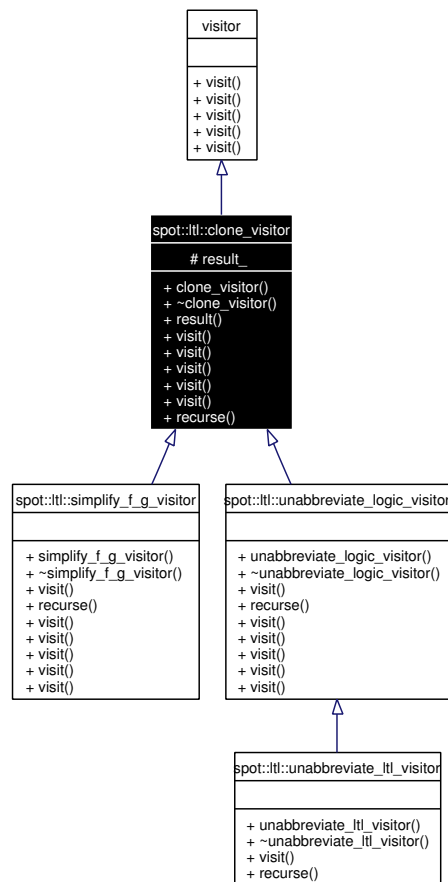
- `ltlast/binop.hh`

**7.7** `spot::ltl::clone_visitor` Class Reference

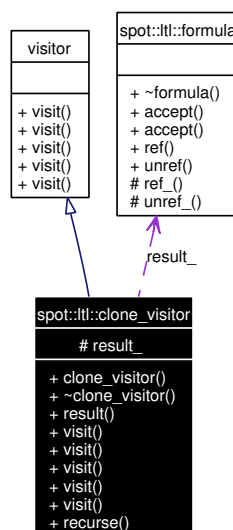
Clone a formula.

```
#include <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\\_](#)

#### 7.7.1 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.

#### 7.7.2 Constructor & Destructor Documentation

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

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

#### 7.7.3 Member Function Documentation

##### 7.7.3.1 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](#).

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

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

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

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

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

##### 7.7.3.5 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](#).

**7.7.3.6** void spot::ltl::clone\_visitor::visit (**unop** \* *uo*) [virtual]

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

Reimplemented in [spot::ltl::unabbreviate\\_ltl\\_visitor](#).

**7.7.3.7** void spot::ltl::clone\_visitor::visit (**atomic\_prop** \* *ap*) [virtual]

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

## 7.7.4 Member Data Documentation

**7.7.4.1** formula\* [spot::ltl::clone\\_visitor::result\\_](#) [protected]

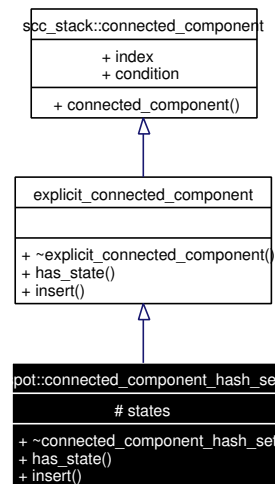
The documentation for this class was generated from the following file:

- ltlvisit/clone.hh

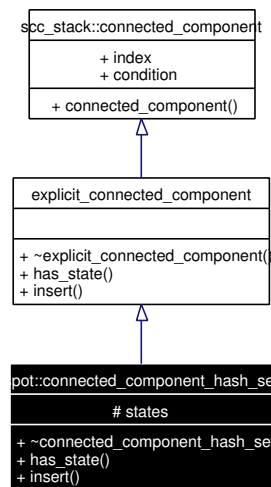
## 7.8 spot::connected\_component\_hash\_set Class Reference

```
#include <explscc.hh>
```

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](#)

### Protected Types

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

### Protected Attributes

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

#### 7.8.1 Detailed Description

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

## 7.8.2 Member Typedef Documentation

**7.8.2.1** typedef Sgi::hash\_set<const [state\\*](#), [state\\_ptr\\_hash](#), [state\\_ptr\\_equal](#)> [spot::connected\\_component\\_hash\\_set::set\\_type](#) [protected]

## 7.8.3 Constructor & Destructor Documentation

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

## 7.8.4 Member Function Documentation

**7.8.4.1** 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](#).

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

Insert a new state in the SCC.

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

## 7.8.5 Member Data Documentation

**7.8.5.1** 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.

**7.8.5.2** int [spot::scc\\_stack::connected\\_component::index](#) [inherited]

Index of the SCC.

**7.8.5.3** [set\\_type](#) [spot::connected\\_component\\_hash\\_set::states](#) [protected]

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

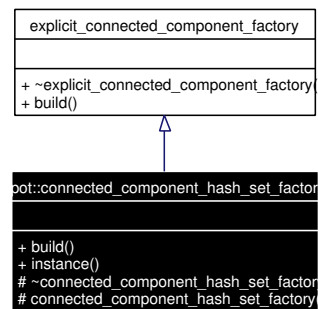
- [tgbaalgos/gtec/explscg.hh](#)

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

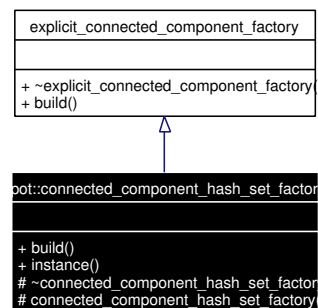
Factory for [connected\\_component\\_hash\\_set](#).

```
#include <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

- 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.

#### 7.9.1 Detailed Description

Factory for `connected_component_hash_set`.

This class is a singleton. Retrieve the instance using `instance()`.

## 7.9.2 Constructor & Destructor Documentation

**7.9.2.1** virtual `spot::connected_component_hash_set_factory::~~connected_component_hash_set_factory()` `[inline, protected, virtual]`

**7.9.2.2** `spot::connected_component_hash_set_factory::connected_component_hash_set_factory()` `[protected]`

Construction is forbidden.

## 7.9.3 Member Function Documentation

**7.9.3.1** 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`.

**7.9.3.2** `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:

- `tgbaalgorithms/gtec/explscs.hh`

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

Formula visitor that cannot modify the formula.

```
#include <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

### 7.10.1 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.

### 7.10.2 Member Function Documentation

**7.10.2.1** virtual void `spot::ltl::const_visitor::visit` (const `multop` \*node) `[pure virtual]`

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

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

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

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

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

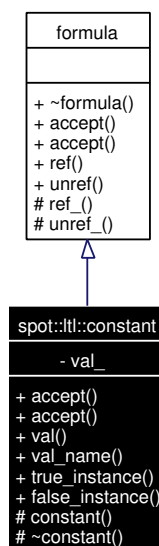
- ltlast/[visitor.hh](#)

## 7.11 spot::ltl::constant Class Reference

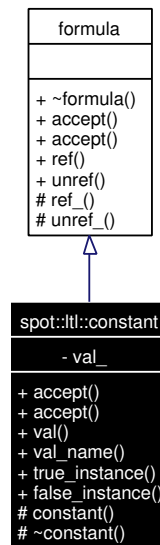
A constant (True or False).

```
#include <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*

## Static Public Member Functions

- `constant` \* `true_instance` ()  
*Get the sole instance of `spot::ltl::constant::constant(True)`.*
- `constant` \* `false_instance` ()  
*Get the sole instance of `spot::ltl::constant::constant(False)`.*



- 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).*

### Private Attributes

- [type](#) [val\\_](#)

#### 7.11.1 Detailed Description

A constant (True or False).

#### 7.11.2 Member Enumeration Documentation

##### 7.11.2.1 enum [spot::ltl::constant::type](#)

Enumeration values:

*False*

*True*

#### 7.11.3 Constructor & Destructor Documentation

##### 7.11.3.1 [spot::ltl::constant::constant](#) ([type](#) val) [protected]

##### 7.11.3.2 virtual [spot::ltl::constant::~~constant](#) () [protected, virtual]

#### 7.11.4 Member Function Documentation

##### 7.11.4.1 virtual void [spot::ltl::constant::accept](#) ([const\\_visitor](#) &v) const [virtual]

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

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

##### 7.11.4.2 virtual void [spot::ltl::constant::accept](#) ([visitor](#) &v) [virtual]

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

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

**7.11.4.3** `constant*` `spot::ltl::constant::false_instance()` `[static]`

Get the sole instance of `spot::ltl::constant::constant(False)`.

**7.11.4.4** `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.

**7.11.4.5** `virtual void` `spot::ltl::formula::ref_()` `[protected, virtual, inherited]`

increment reference counter if any

Reimplemented in `spot::ltl::ref_formula`.

**7.11.4.6** `constant*` `spot::ltl::constant::true_instance()` `[static]`

Get the sole instance of `spot::ltl::constant::constant(True)`.

**7.11.4.7** `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.

**7.11.4.8** `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`.

**7.11.4.9** `type` `spot::ltl::constant::val()` `const`

Return the value of the constant.

**7.11.4.10** `const char*` `spot::ltl::constant::val_name()` `const`

Return the value of the constant as a string.

**7.11.5 Member Data Documentation****7.11.5.1** `type` `spot::ltl::constant::val_` `[private]`

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

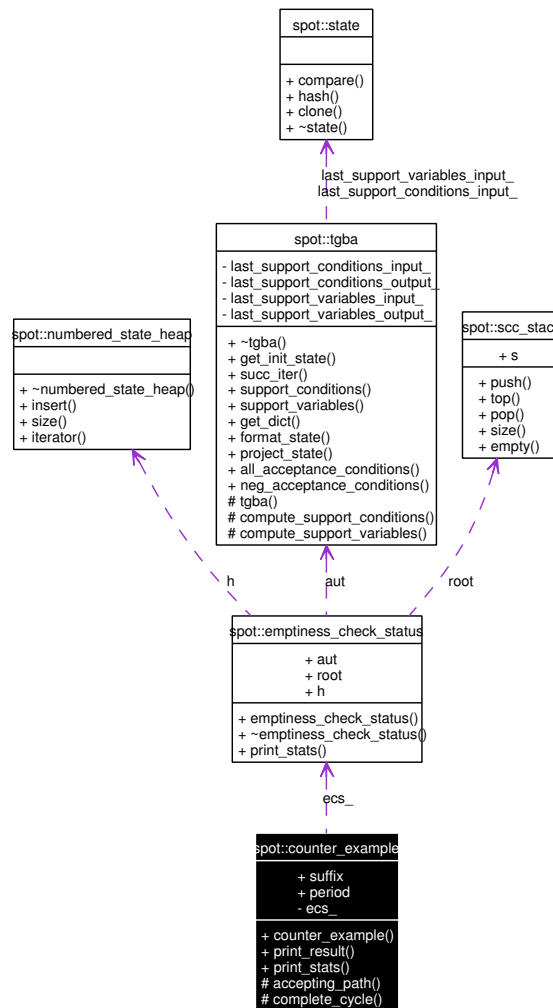
- `ltlast/constant.hh`

## 7.12 spot::counter\_example Class Reference

Compute a counter example from a [spot::emptiness\\_check\\_status](#).

```
#include <ce.hh>
```

Collaboration diagram for spot::counter\_example:



### Public Types

- typedef std::pair< const [state](#) \*, bdd > [state\\_proposition](#)
- typedef std::list< const [state](#) \* > [state\\_sequence](#)
- typedef std::list< [state\\_proposition](#) > [cycle\\_path](#)

### Public Member Functions

- [counter\\_example](#) (const [emptiness\\_check\\_status](#) \*ecs, const [explicit\\_connected\\_component\\_factory](#) \*eccf=connected\_component\_hash\_set\_factory::instance())
- std::ostream & [print\\_result](#) (std::ostream &os, const [tgba](#) \*restrict=0) const

*Display the example computed by counter\_example().*

- void [print\\_stats](#) (std::ostream &os) const

*Output statistics about this object.*

### Public Attributes

- [state\\_sequence](#) suffix
- [cycle\\_path](#) period

### Protected Member Functions

- void [accepting\\_path](#) (const [explicit\\_connected\\_component](#) \*scc, const [state](#) \*start, bdd acc\_to\_traverse)
- void [complete\\_cycle](#) (const [explicit\\_connected\\_component](#) \*scc, const [state](#) \*from, const [state](#) \*to)

### Private Attributes

- const [emptiness\\_check\\_status](#) \*ecs\_

#### 7.12.1 Detailed Description

Compute a counter example from a [spot::emptiness\\_check\\_status](#).

#### 7.12.2 Member Typedef Documentation

7.12.2.1 typedef std::list<[state\\_proposition](#)> [spot::counter\\_example::cycle\\_path](#)

7.12.2.2 typedef std::pair<const [state](#)\*, bdd> [spot::counter\\_example::state\\_proposition](#)

7.12.2.3 typedef std::list<const [state](#)\*> [spot::counter\\_example::state\\_sequence](#)

#### 7.12.3 Constructor & Destructor Documentation

7.12.3.1 [spot::counter\\_example::counter\\_example](#) (const [emptiness\\_check\\_status](#) \*ecs, const [explicit\\_connected\\_component\\_factory](#) \*eccf = connected\_component\_hash\_set\_factory::instance())

#### 7.12.4 Member Function Documentation

7.12.4.1 void [spot::counter\\_example::accepting\\_path](#) (const [explicit\\_connected\\_component](#) \*scc, const [state](#) \*start, bdd acc\_to\_traverse) [protected]

Called by [counter\\_example](#) to find a path which traverses all acceptance conditions in the accepted SCC.

**7.12.4.2** void spot::counter\_example::complete\_cycle (const explicit\_connected\_component \* *scc*, const state \* *from*, const state \* *to*) [protected]

Complete a cycle that caraterise the period of the counter example. Append a sequence to the path given by accepting\_path.

**7.12.4.3** std::ostream& spot::counter\_example::print\_result (std::ostream & *os*, const tgba \* *restrict* = 0) const

Display the example computed by counter\_example().

#### Parameters:

*os* the output stream

*restrict* optional automaton to project the example on.

**7.12.4.4** void spot::counter\_example::print\_stats (std::ostream & *os*) const

Output statistics about this object.

### 7.12.5 Member Data Documentation

**7.12.5.1** const emptiness\_check\_status\* spot::counter\_example::ecs\_ [private]

**7.12.5.2** cycle\_path spot::counter\_example::period

**7.12.5.3** state\_sequence spot::counter\_example::suffix

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

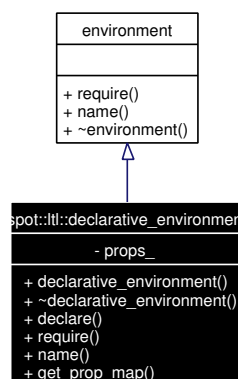
- [tgbaalgorithms/gtec/ce.hh](#)

## 7.13 spot::ltl::declarative\_environment Class Reference

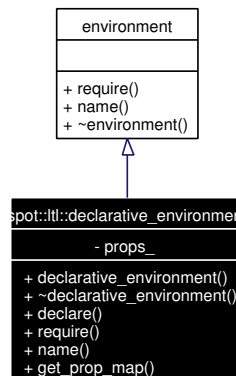
A declarative environment.

```
#include <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\_

### 7.13.1 Detailed Description

A declarative environment.

This environment recognizes all atomic propositions that have been previously declared. It will reject other.

### 7.13.2 Member Typedef Documentation

**7.13.2.1** typedef std::map<const std::string, [ltl::atomic\\_prop](#)\*> [spot::ltl::declarative\\_environment::prop\\_map](#)

### 7.13.3 Constructor & Destructor Documentation

**7.13.3.1** `spot::ltl::declarative_environment::declarative_environment ()`

**7.13.3.2** `spot::ltl::declarative_environment::~~declarative_environment ()`

### 7.13.4 Member Function Documentation

**7.13.4.1** `bool spot::ltl::declarative_environment::declare (const std::string & prop_str)`

Declare an atomic proposition. Return false iff the proposition was already declared.

**7.13.4.2** `const prop\_map & spot::ltl::declarative_environment::get_prop_map () const`

Get the map of atomic proposition known to this environment.

**7.13.4.3** `virtual const std::string & spot::ltl::declarative_environment::name () [virtual]`

Get the name of the environment.

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

**7.13.4.4** `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](#).

### 7.13.5 Member Data Documentation

**7.13.5.1** `prop\_map spot::ltl::declarative_environment::props_ [private]`

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

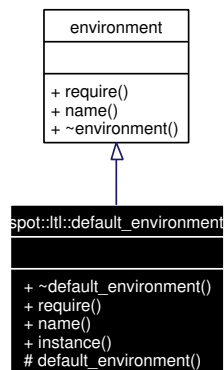
- [ltlenv/declenv.hh](#)

## 7.14 `spot::ltl::default_environment` Class Reference

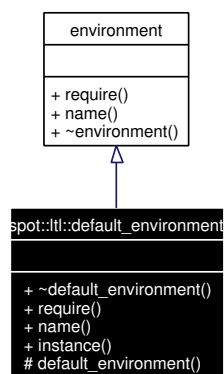
A laxist environment.

```
#include <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

- `default_environment & instance()`  
*Get the sole instance of spot::ltl::default\_environment.*

### Protected Member Functions

- `default_environment()`



### 7.14.1 Detailed Description

A laxist environment.

This environment recognizes all atomic propositions.

This is a singleton. Use `default_environment::instance()` to obtain the instance.

### 7.14.2 Constructor & Destructor Documentation

**7.14.2.1** `virtual spot::ltl::default_environment::~~default_environment() [virtual]`

**7.14.2.2** `spot::ltl::default_environment::default_environment() [protected]`

### 7.14.3 Member Function Documentation

**7.14.3.1** `default_environment& spot::ltl::default_environment::instance() [static]`

Get the sole instance of `spot::ltl::default_environment`.

**7.14.3.2** `virtual const std::string& spot::ltl::default_environment::name() [virtual]`

Get the name of the environment.

Implements `spot::ltl::environment`.

**7.14.3.3** `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:

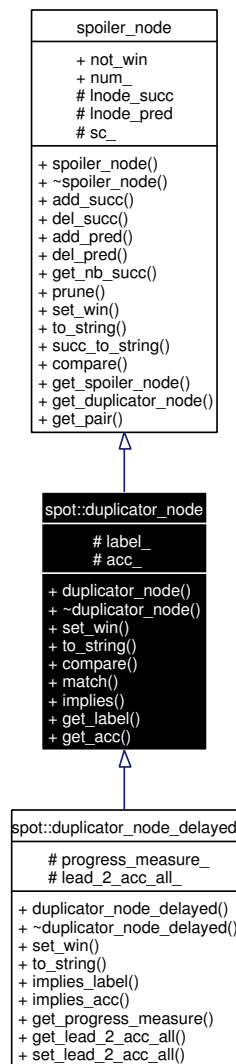
- `ltenv/defaultenv.hh`

## 7.15 `spot::duplicator_node` Class Reference

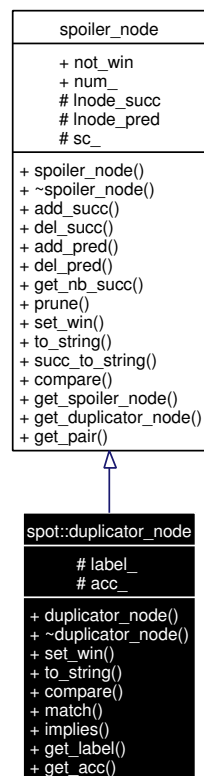
Duplicator node of parity game graph.

```
#include <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\\_](#)

**7.15.1 Detailed Description**

Duplicator node of parity game graph.

**7.15.2 Constructor & Destructor Documentation**

**7.15.2.1** `spot::duplicator_node::duplicator_node (const state * d_node, const state * s_node, bdd l, bdd a, int num)`

**7.15.2.2** `virtual spot::duplicator_node::~~duplicator_node ()` [virtual]

**7.15.3 Member Function Documentation**

**7.15.3.1** `virtual void spot::spoiler_node::add_pred (spoiler\_node * n)` [virtual, inherited]

**7.15.3.2** `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.

**7.15.3.3** `virtual bool spot::duplicator_node::compare (spoiler\_node * n)` [virtual]

Reimplemented from [spot::spoiler\\_node](#).

**7.15.3.4** `virtual void spot::spoiler_node::del_pred ()` [virtual, inherited]

**7.15.3.5** `void spot::spoiler_node::del_succ (spoiler\_node * n)` [inherited]

**7.15.3.6** `bdd spot::duplicator_node::get_acc () const`

**7.15.3.7** `const state* spot::spoiler_node::get_duplicator_node ()` [inherited]

**7.15.3.8** `bdd spot::duplicator_node::get_label () const`

**7.15.3.9** `int spot::spoiler_node::get_nb_succ ()` [inherited]

**7.15.3.10** `state_couple* spot::spoiler_node::get_pair ()` [inherited]

**7.15.3.11** `const state* spot::spoiler_node::get_spoiler_node ()` [inherited]

**7.15.3.12** `bool spot::duplicator_node::implies (bdd l, bdd a)`

**7.15.3.13** `bool spot::duplicator_node::match (bdd l, bdd a)`

**7.15.3.14** `bool spot::spoiler_node::prune ()` [inherited]

**7.15.3.15** `virtual bool spot::duplicator_node::set_win ()` [virtual]

Reimplemented from `spot::spoiler_node`.

Reimplemented in `spot::duplicator_node_delayed`.

**7.15.3.16** `virtual std::string spot::spoiler_node::succ_to_string ()` [virtual, inherited]

**7.15.3.17** `virtual std::string spot::duplicator_node::to_string (const tgba * a)` [virtual]

Reimplemented from `spot::spoiler_node`.

Reimplemented in `spot::duplicator_node_delayed`.

## 7.15.4 Member Data Documentation

**7.15.4.1** `bdd spot::duplicator_node::acc_` [protected]

**7.15.4.2** `bdd spot::duplicator_node::label_` [protected]

**7.15.4.3** `sn_v* spot::spoiler_node::lnode_pred` [protected, inherited]

**7.15.4.4** `sn_v* spot::spoiler_node::lnode_succ` [protected, inherited]

**7.15.4.5** `bool spot::spoiler_node::not_win` [inherited]

**7.15.4.6** `int spot::spoiler_node::num_` [inherited]

**7.15.4.7** `state_couple* spot::spoiler_node::sc_` [protected, inherited]

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

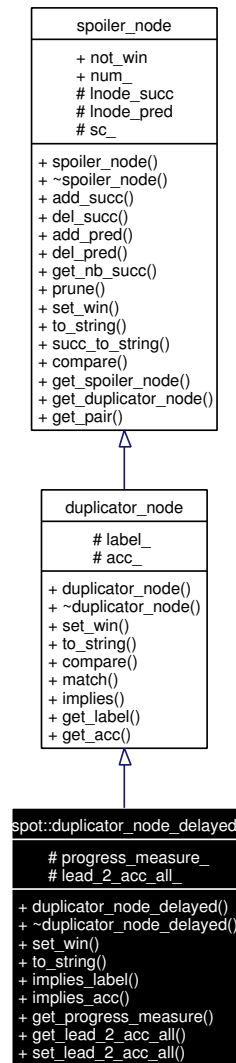
- `tgbaalgos/reductgba_sim.hh`

## 7.16 spot::duplicator\_node\_delayed Class Reference

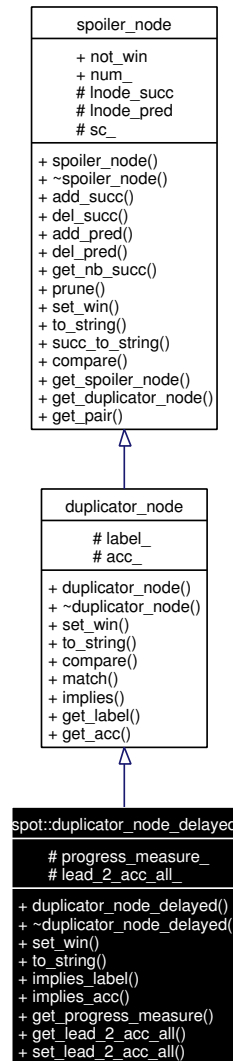
Duplicator node of parity game graph for delayed simulation.

```
#include <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](#) ()
- void [set\\_lead\\_2\\_acc\\_all](#) ()
- 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

- int [progress\\_measure\\_](#)
- bool [lead\\_2\\_acc\\_all\\_](#)
- bdd [label\\_](#)
- bdd [acc\\_](#)
- sn\_v \* [lnode\\_succ](#)
- sn\_v \* [lnode\\_pred](#)
- state\_couple \* [sc\\_](#)

#### 7.16.1 Detailed Description

Duplicator node of parity game graph for delayed simulation.

#### 7.16.2 Constructor & Destructor Documentation

**7.16.2.1** spot::duplicator\_node\_delayed::duplicator\_node\_delayed (const state \* [d\\_node](#), const state \* [s\\_node](#), bdd [l](#), bdd [a](#), int [num](#))

**7.16.2.2** spot::duplicator\_node\_delayed::~~duplicator\_node\_delayed ()

#### 7.16.3 Member Function Documentation

**7.16.3.1** virtual void spot::spoiler\_node::add\_pred (spoiler\_node \* [n](#)) [virtual, inherited]

**7.16.3.2** 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.



**7.16.3.3** `virtual bool spot::duplicator_node::compare (spoiler_node * n) [virtual, inherited]`

Reimplemented from [spot::spoiler\\_node](#).

**7.16.3.4** `virtual void spot::spoiler_node::del_pred () [virtual, inherited]`

**7.16.3.5** `void spot::spoiler_node::del_succ (spoiler_node * n) [inherited]`

**7.16.3.6** `bdd spot::duplicator_node::get_acc () const [inherited]`

**7.16.3.7** `const state* spot::spoiler_node::get_duplicator_node () [inherited]`

**7.16.3.8** `bdd spot::duplicator_node::get_label () const [inherited]`

**7.16.3.9** `bool spot::duplicator_node_delayed::get_lead_2_acc_all ()`

**7.16.3.10** `int spot::spoiler_node::get_nb_succ () [inherited]`

**7.16.3.11** `state_couple* spot::spoiler_node::get_pair () [inherited]`

**7.16.3.12** `int spot::duplicator_node_delayed::get_progress_measure ()`

**7.16.3.13** `const state* spot::spoiler_node::get_spoiler_node () [inherited]`

**7.16.3.14** `bool spot::duplicator_node::implies (bdd l, bdd a) [inherited]`

**7.16.3.15** `bool spot::duplicator_node_delayed::implies_acc (bdd a)`

**7.16.3.16** `bool spot::duplicator_node_delayed::implies_label (bdd l)`

**7.16.3.17** `bool spot::duplicator_node::match (bdd l, bdd a) [inherited]`

**7.16.3.18** `bool spot::spoiler_node::prune () [inherited]`

**7.16.3.19** `void spot::duplicator_node_delayed::set_lead_2_acc_all ()`

**7.16.3.20** `bool spot::duplicator_node_delayed::set_win () [virtual]`

Return true if the progress\_measure has changed.

Reimplemented from [spot::duplicator\\_node](#).

**7.16.3.21** virtual std::string spot::spoiler\_node::succ\_to\_string () [virtual, inherited]

**7.16.3.22** virtual std::string spot::duplicator\_node\_delayed::to\_string (const tgba \* a) [virtual]

Reimplemented from spot::duplicator\_node.

## 7.16.4 Member Data Documentation

**7.16.4.1** bdd spot::duplicator\_node::acc\_ [protected, inherited]

**7.16.4.2** bdd spot::duplicator\_node::label\_ [protected, inherited]

**7.16.4.3** bool spot::duplicator\_node\_delayed::lead\_2\_acc\_all\_ [protected]

**7.16.4.4** sn\_v\* spot::spoiler\_node::lnode\_pred [protected, inherited]

**7.16.4.5** sn\_v\* spot::spoiler\_node::lnode\_succ [protected, inherited]

**7.16.4.6** bool spot::spoiler\_node::not\_win [inherited]

**7.16.4.7** int spot::spoiler\_node::num\_ [inherited]

**7.16.4.8** int spot::duplicator\_node\_delayed::progress\_measure\_ [protected]

**7.16.4.9** state\_couple\* spot::spoiler\_node::sc\_ [protected, inherited]

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

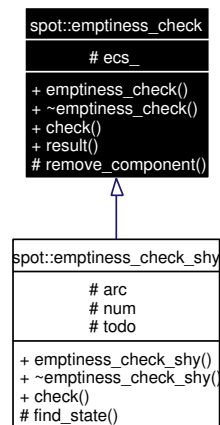
- tgbalgorithms/reductgba\_sim.hh

## 7.17 spot::emptiness\_check Class Reference

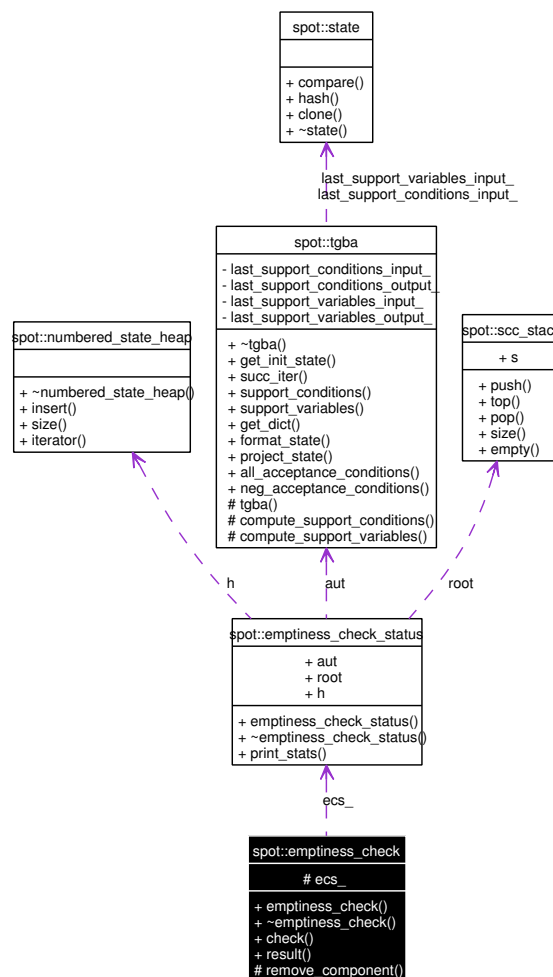
Check whether the language of an automate is empty.

```
#include <gttec.hh>
```

Inheritance diagram for spot::emptiness\_check:



Collaboration diagram for `spot::emptiness_check`:



**Public Member Functions**

- `emptiness_check` (const `tgba` \*a, const `numbered_state_heap_factory` \*nshf=numbered\_state\_heap\_factory::instance())
- virtual `~emptiness_check` ()
- virtual bool `check` ()  
*Check whether the automaton's language is empty.*
- const `emptiness_check_status` \* `result` () const  
*Return the status of the emptiness-check.*

**Protected Member Functions**

- void `remove_component` (const `state` \*start\_delete)  
*Remove a strongly component from the hash.*

**Protected Attributes**

- `emptiness_check_status` \* `ecs_`

**7.17.1 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::emptiness_check` and `spot::emptiness_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::emptiness_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::emptiness_check_shy` try 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.

### 7.17.2 Constructor & Destructor Documentation

**7.17.2.1** `spot::emptiness_check::emptiness_check (const tgba * a, const numbered\_state\_heap\_factory * nshf = numbered_state_heap_hash_map_factory::instance())`

**7.17.2.2** `virtual spot::emptiness_check::~emptiness_check ()` [virtual]

### 7.17.3 Member Function Documentation

**7.17.3.1** `virtual bool spot::emptiness_check::check ()` [virtual]

Check whether the automaton's language is empty.

Reimplemented in [spot::emptiness\\_check\\_shy](#).

**7.17.3.2** `void spot::emptiness_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.

**7.17.3.3** `const emptiness\_check\_status* spot::emptiness_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 [emptiness\\_check](#) object is deleted.

### 7.17.4 Member Data Documentation

**7.17.4.1** `emptiness\_check\_status* spot::emptiness_check::ecs_` [protected]

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

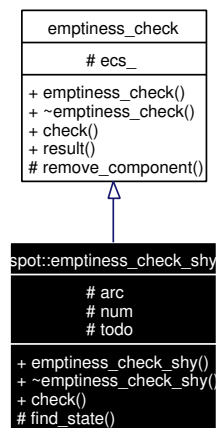
- [tgbaalgos/gtec/gtec.hh](#)

## 7.18 spot::emptiness\_check\_shy Class Reference

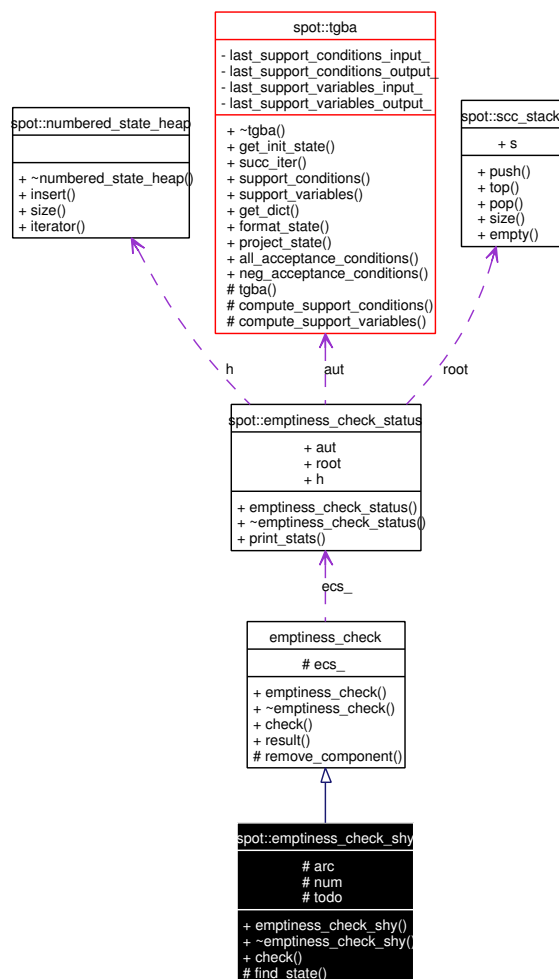
A version of [spot::emptiness\\_check](#) try to visit known states first.

```
#include <gtec.hh>
```

Inheritance diagram for `spot::emptiness_check_shy`:



Collaboration diagram for `spot::emptiness_check_shy`:



### Public Member Functions

- `emptiness_check_shy` (const `tgba` \*a, const `numbered_state_heap_factory` \*nshf=numbered\_state\_heap\_hash\_map\_factory::instance())
- virtual `~emptiness_check_shy` ()
- virtual bool `check` ()  
*Check whether the automaton's language is empty.*
- const `emptiness_check_status` \* `result` () const  
*Return the status of the emptiness-check.*

### Protected Types

- typedef std::list< `successor` > `succ_queue`
- typedef std::pair< const `state` \*, `succ_queue` > `pair_state_successors`

### Protected Member Functions

- virtual int \* `find_state` (const `state` \*s)
- void `remove_component` (const `state` \*start\_delete)  
*Remove a strongly component from the hash.*

### Protected Attributes

- std::stack< bdd > `arc`
- int `num`
- std::stack< `pair_state_successors` > `todo`
- `emptiness_check_status` \* `ecs_`

#### 7.18.1 Detailed Description

A version of `spot::emptiness_check` try to visit known states first.

See the documentation for `spot::emptiness_check`

#### 7.18.2 Member Typedef Documentation

**7.18.2.1** typedef std::pair<const `state`\*, `succ_queue`> `spot::emptiness_check_shy::pair_state_successors` [protected]

**7.18.2.2** typedef std::list<`successor`> `spot::emptiness_check_shy::succ_queue` [protected]

#### 7.18.3 Constructor & Destructor Documentation

**7.18.3.1** `spot::emptiness_check_shy::emptiness_check_shy` (const `tgba` \*a, const `numbered_state_heap_factory` \*nshf=numbered\_state\_heap\_hash\_map\_factory::instance())

**7.18.3.2** `virtual spot::emptiness_check_shy::~~emptiness_check_shy ()` [virtual]

#### 7.18.4 Member Function Documentation

**7.18.4.1** `virtual bool spot::emptiness_check_shy::check ()` [virtual]

Check whether the automaton's language is empty.

Reimplemented from [spot::emptiness\\_check](#).

**7.18.4.2** `virtual int* spot::emptiness_check_shy::find_state (const state * s)` [protected, virtual]

**7.18.4.3** `void spot::emptiness_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.

**7.18.4.4** `const emptiness_check_status* spot::emptiness_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 [emptiness\\_check](#) object is deleted.

#### 7.18.5 Member Data Documentation

**7.18.5.1** `std::stack<bdd> spot::emptiness_check_shy::arc` [protected]

**7.18.5.2** `emptiness_check_status* spot::emptiness_check::ecs_` [protected, inherited]

**7.18.5.3** `int spot::emptiness_check_shy::num` [protected]

**7.18.5.4** `std::stack<pair_state_successors> spot::emptiness_check_shy::todo` [protected]

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

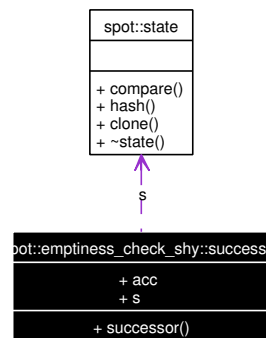
- [tgbaalgos/gtec/gtec.hh](#)

## 7.19 spot::emptiness\_check\_shy::successor Struct Reference

```
#include <gtec.hh>
```

Collaboration diagram for `spot::emptiness_check_shy::successor`:





### Public Member Functions

- [successor](#) (bdd [acc](#), const [spot::state](#) \*[s](#))

### Public Attributes

- bdd [acc](#)
- const [spot::state](#) \* [s](#)

#### 7.19.1 Constructor & Destructor Documentation

**7.19.1.1** [spot::emptiness\\_check\\_shy::successor::successor](#) (bdd [acc](#), const [spot::state](#) \* [s](#))  
[inline]

#### 7.19.2 Member Data Documentation

**7.19.2.1** bdd [spot::emptiness\\_check\\_shy::successor::acc](#)

**7.19.2.2** const [spot::state](#)\* [spot::emptiness\\_check\\_shy::successor::s](#)

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

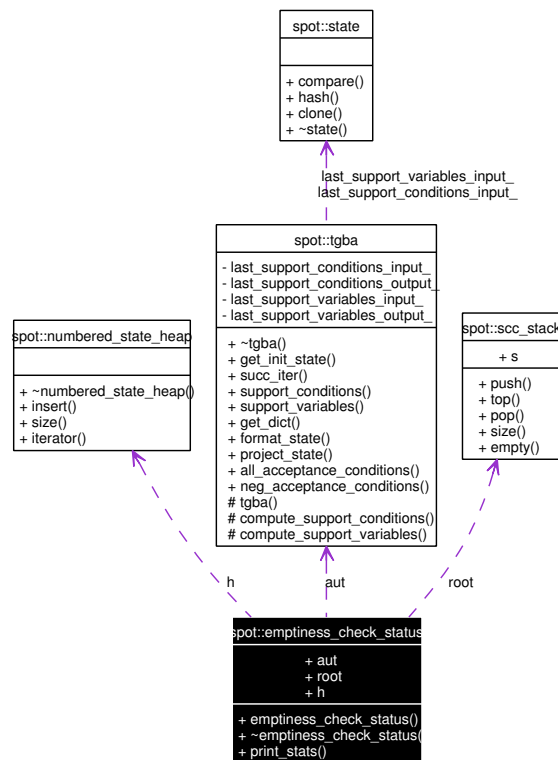
- [tgbaalgos/gtec/gtec.hh](#)

## 7.20 spot::emptiness\_check\_status Class Reference

The status of the emptiness-check on success.

```
#include <status.hh>
```

Collaboration diagram for [spot::emptiness\\_check\\_status](#):



## Public Member Functions

- `emptiness_check_status` (const `tgba` \*`aut`, const `numbered_state_heap_factory` \*`nshf`)
- `~emptiness_check_status` ()
- void `print_stats` (std::ostream &`os`) const

*Output statistics about this object.*

## Public Attributes

- const `tgba` \* `aut`
- `scc_stack` `root`
- `numbered_state_heap` \* `h`

*Heap of visited states.*

### 7.20.1 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.

## 7.20.2 Constructor & Destructor Documentation

**7.20.2.1** `spot::emptiness_check_status::emptiness_check_status (const tgba * aut, const numbered\_state\_heap\_factory * nshf)`

**7.20.2.2** `spot::emptiness_check_status::~~emptiness_check_status ()`

## 7.20.3 Member Function Documentation

**7.20.3.1** `void spot::emptiness_check_status::print_stats (std::ostream & os) const`

Output statistics about this object.

## 7.20.4 Member Data Documentation

**7.20.4.1** `const tgba* spot::emptiness_check_status::aut`

**7.20.4.2** `numbered\_state\_heap* spot::emptiness_check_status::h`

Heap of visited states.

**7.20.4.3** `scc\_stack spot::emptiness_check_status::root`

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

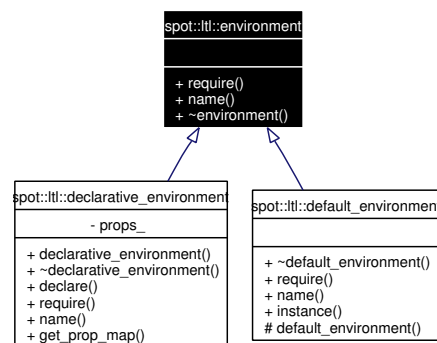
- [tgbaalgos/gtec/status.hh](#)

## 7.21 spot::ltl::environment Class Reference

An environment that describes atomic propositions.

```
#include <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` ()

### 7.21.1 Detailed Description

An environment that describes atomic propositions.

### 7.21.2 Constructor & Destructor Documentation

**7.21.2.1** virtual `spot::ltl::environment::~~environment` () [inline, virtual]

### 7.21.3 Member Function Documentation

**7.21.3.1** 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`.

**7.21.3.2** 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:

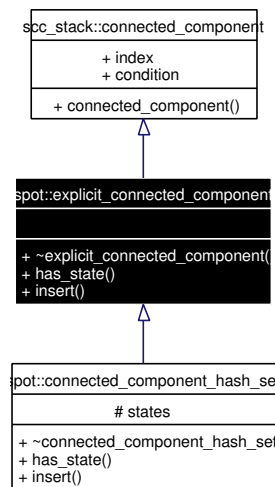
- `ltlenv/environment.hh`

## 7.22 `spot::explicit_connected_component` Class Reference

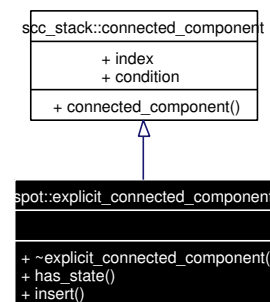
An SCC storing all its states explicitly.

```
#include <explscc.hh>
```

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`

### 7.22.1 Detailed Description

An SCC storing all its states explicitly.

### 7.22.2 Constructor & Destructor Documentation

**7.22.2.1** `virtual spot::explicit_connected_component::~explicit_connected_component ()`  
[inline, virtual]

### 7.22.3 Member Function Documentation

**7.22.3.1** `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`.

**7.22.3.2** `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`.

### 7.22.4 Member Data Documentation

**7.22.4.1** `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.

**7.22.4.2** `int spot::scc_stack::connected_component::index` [inherited]

Index of the SCC.

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

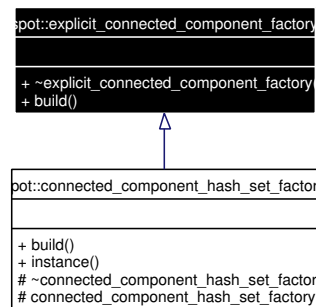
- `tgbaalgos/gtec/explscs.hh`

## 7.23 `spot::explicit_connected_component_factory` Class Reference

Abstract factory for `explicit_connected_component`.

```
#include <explscs.hh>
```

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](#).

#### 7.23.1 Detailed Description

Abstract factory for [explicit\\_connected\\_component](#).

#### 7.23.2 Constructor & Destructor Documentation

**7.23.2.1** virtual [spot::explicit\\_connected\\_component\\_factory::~explicit\\_connected\\_component\\_factory](#) () [inline, virtual]

#### 7.23.3 Member Function Documentation

**7.23.3.1** 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:

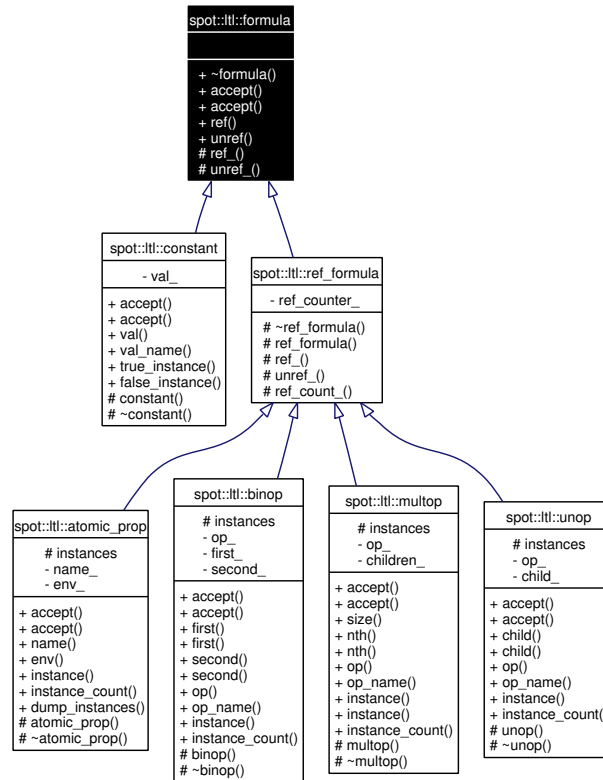
- [tgbaalgos/gtec/explsc.hh](#)

## 7.24 spot::ltl::formula Class Reference

An LTL formula.

```
#include <formula.hh>
```

Inheritance diagram for [spot::ltl::formula](#):



## Public Member Functions

- virtual `~formula()`
- 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

## Static Public Member Functions

- void `unref(formula *f)`  
release this node

## Protected Member Functions

- 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).*

### 7.24.1 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](#).

### 7.24.2 Constructor & Destructor Documentation

#### 7.24.2.1 virtual spot::ltl::formula::~~formula() [virtual]

### 7.24.3 Member Function Documentation

#### 7.24.3.1 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](#).

#### 7.24.3.2 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](#).

#### 7.24.3.3 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.

#### 7.24.3.4 virtual void spot::ltl::formula::ref\_() [protected, virtual]

increment reference counter if any

Reimplemented in [spot::ltl::ref\\_formula](#).

#### 7.24.3.5 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.

### 7.24.3.6 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](#).

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

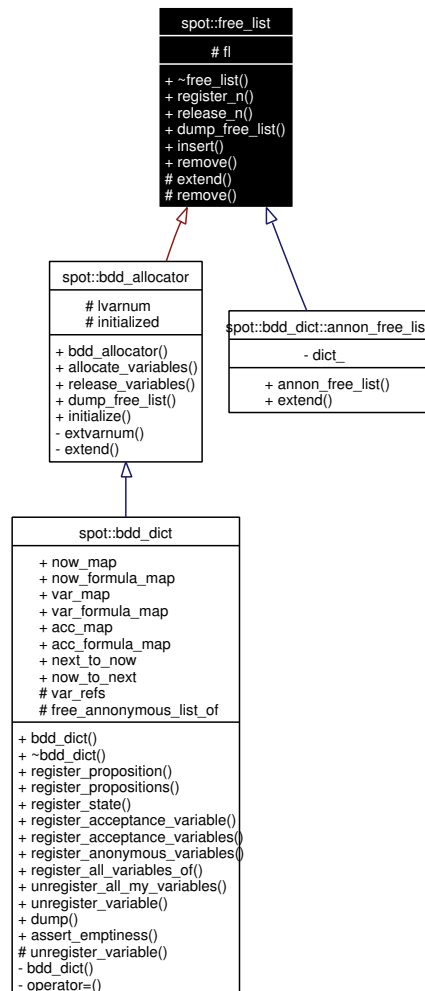
- [ltlast/formula.hh](#)

## 7.25 spot::free\_list Class Reference

Manage list of free integers.

```
#include <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.*

#### 7.25.1 Detailed Description

Manage list of free integers.

#### 7.25.2 Member Typedef Documentation

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

### 7.25.2.2 `typedef std::pair<int, int> spot::free_list::pos_lenght_pair` [protected]

Such pairs describe `second` free integer starting at `first`.

## 7.25.3 Constructor & Destructor Documentation

### 7.25.3.1 `virtual spot::free_list::~~free_list()` [virtual]

## 7.25.4 Member Function Documentation

### 7.25.4.1 `std::ostream& spot::free_list::dump_free_list(std::ostream & os) const`

Dump the list to `os` for debugging.

### 7.25.4.2 `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`.

### 7.25.4.3 `void spot::free_list::insert(int base, int n)`

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

### 7.25.4.4 `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

### 7.25.4.5 `void spot::free_list::release_n(int base, int n)`

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

### 7.25.4.6 `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`.

### 7.25.4.7 `void spot::free_list::remove(int base, int n = 0)`

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

### 7.25.5 Member Data Documentation

#### 7.25.5.1 [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](#)

## 7.26 spot::gspn\_exception Class Reference

An exeption used to forward GSPN errors.

```
#include <common.hh>
```

### Public Member Functions

- [gspn\\_exception](#) (const std::string &where, int err)
- int [get\\_err](#) () const
- std::string [get\\_where](#) () const

### Private Attributes

- int [err\\_](#)
- std::string [where\\_](#)

### 7.26.1 Detailed Description

An exeption used to forward GSPN errors.

### 7.26.2 Constructor & Destructor Documentation

#### 7.26.2.1 [spot::gspn\\_exception::gspn\\_exception](#) (const std::string & *where*, int *err*) [inline]

### 7.26.3 Member Function Documentation

#### 7.26.3.1 int [spot::gspn\\_exception::get\\_err](#) () const [inline]

#### 7.26.3.2 std::string [spot::gspn\\_exception::get\\_where](#) () const [inline]

### 7.26.4 Member Data Documentation

#### 7.26.4.1 int [spot::gspn\\_exception::err\\_](#) [private]

#### 7.26.4.2 std::string [spot::gspn\\_exception::where\\_](#) [private]

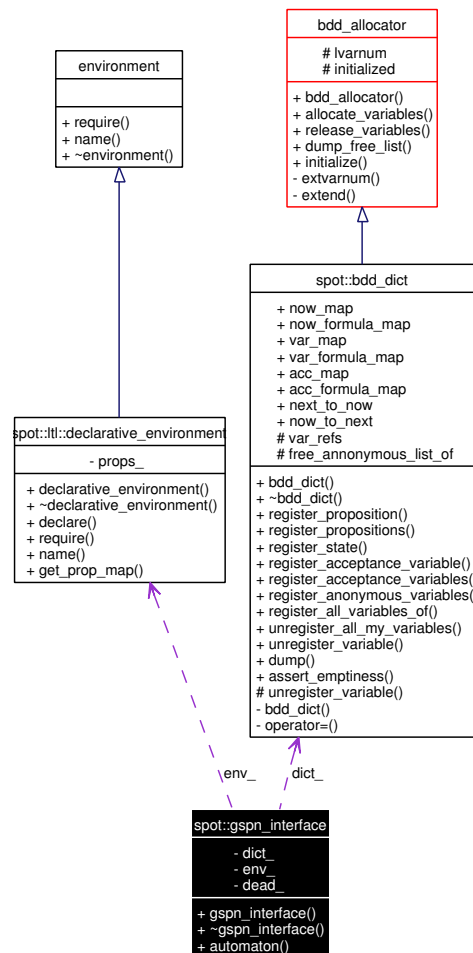
The documentation for this class was generated from the following file:

- [gspn/common.hh](#)

## 7.27 spot::gspn\_interface Class Reference

```
#include <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_`

### 7.27.1 Constructor & Destructor Documentation

**7.27.1.1** `spot::gspn_interface::gspn_interface (int argc, char ** argv, bdd\_dict * dict, ltl::declarative\_environment & env, const std::string & dead = "true")`

**7.27.1.2** `spot::gspn_interface::~gspn_interface ()`

### 7.27.2 Member Function Documentation

**7.27.2.1** `tgba* spot::gspn_interface::automaton () const`

### 7.27.3 Member Data Documentation

**7.27.3.1** `const std::string spot::gspn\_interface::dead\_ [private]`

**7.27.3.2** `bdd\_dict* spot::gspn\_interface::dict\_ [private]`

**7.27.3.3** `ltl::declarative\_environment& spot::gspn\_interface::env\_ [private]`

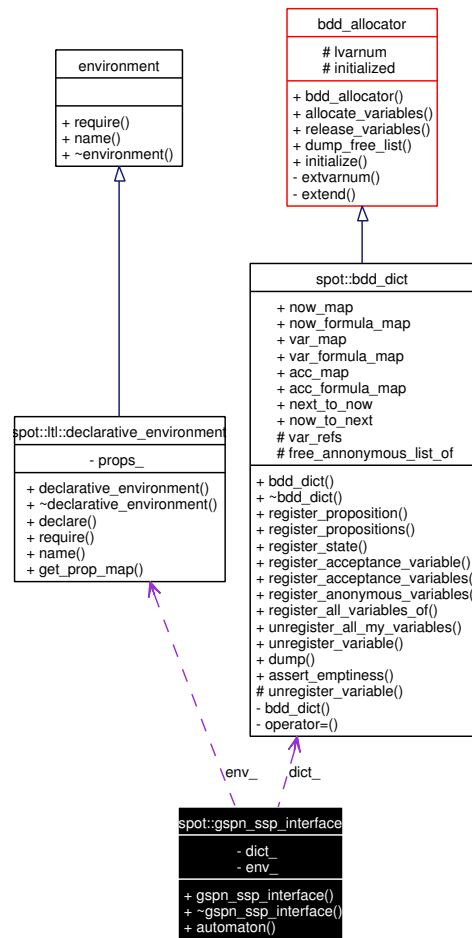
The documentation for this class was generated from the following file:

- [gspn/gspn.hh](#)

## 7.28 spot::gspn\_ssp\_interface Class Reference

```
#include <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\\_](#)

### 7.28.1 Constructor & Destructor Documentation

**7.28.1.1** `spot::gspn_ssp_interface::gspn_ssp_interface (int argc, char ** argv, bdd\_dict * dict, const ltl::declarative\_environment & env, bool inclusion = false)`

**7.28.1.2** `spot::gspn_ssp_interface::~~gspn_ssp_interface ()`



## 7.28.2 Member Function Documentation

7.28.2.1 [tgba\\*](#) [spot::gspn\\_ssp\\_interface::automaton](#) (const [tgba](#) \* *operand*) const

## 7.28.3 Member Data Documentation

7.28.3.1 [bdd\\_dict\\*](#) [spot::gspn\\_ssp\\_interface::dict\\_](#) [private]

7.28.3.2 const [ltl::declarative\\_environment&](#) [spot::gspn\\_ssp\\_interface::env\\_](#) [private]

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

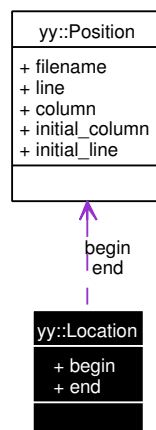
- [gspn/ssp.hh](#)

## 7.29 yy::Location Class Reference

Abstract a [Location](#).

```
#include <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.*

## 7.29.1 Detailed Description

Abstract a [Location](#).

## 7.29.2 Constructor & Destructor Documentation

### 7.29.2.1 yy::Location::Location (void) [inline]

Construct a [Location](#).

## 7.29.3 Member Function Documentation

### 7.29.3.1 void yy::Location::columns (unsigned int count = 1) [inline]

Extend the current location to the COUNT next columns.

### 7.29.3.2 void yy::Location::lines (unsigned int count = 1) [inline]

Extend the current location to the COUNT next lines.

### 7.29.3.3 void yy::Location::step (void) [inline]

Reset initial location to final location.

## 7.29.4 Member Data Documentation

### 7.29.4.1 [Position yy::Location::begin](#)

Beginning of the located region.

### 7.29.4.2 [Position yy::Location::end](#)

End of the located region.

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

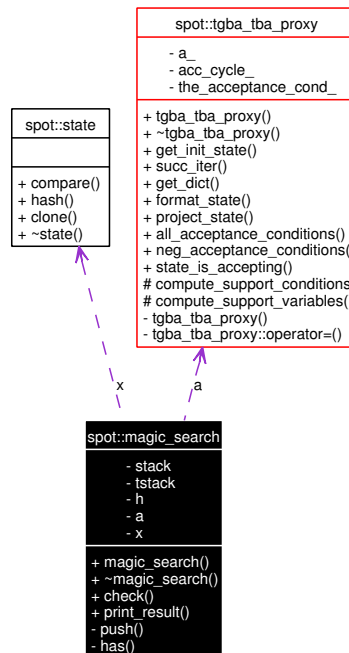
- [ltparse/location.hh](#)

## 7.30 spot::magic\_search Struct Reference

Emptiness check on [spot::tgba\\_tba\\_proxy](#) automata using the Magic Search algorithm.

```
#include <magic.hh>
```

Collaboration diagram for spot::magic\_search:



### Public Member Functions

- [magic\\_search](#) (const [tgba\\_tba\\_proxy](#) \*a)  
*Initialize the Magic Search algorithm on the automaton a.*
- [~magic\\_search](#) ()
- bool [check](#) ()  
*Perform a Magic Search.*
- std::ostream & [print\\_result](#) (std::ostream &os, const [tgba](#) \*restrict=0) const  
*Print the last accepting path found.*

### Private Types

- typedef std::pair< [magic\\_state](#), [tgba\\_succ\\_iterator](#) \* > [state\\_iter\\_pair](#)
- typedef std::list< [state\\_iter\\_pair](#) > [stack\\_type](#)
- typedef std::list< bdd > [tstack\\_type](#)
- typedef Sgi::hash\_map< const [state](#) \*, [magic](#), [state\\_ptr\\_hash](#), [state\\_ptr\\_equal](#) > [hash\\_type](#)

### Private Member Functions

- void [push](#) (const [state](#) \*s, bool m)  
*Append a new state to the current path.*
- bool [has](#) (const [state](#) \*s, bool m) const  
*Check whether we already visited s with the Magic bit set to m.*

### Private Attributes

- [stack\\_type](#) [stack](#)  
*Stack of visited states on the path.*
- [tstack\\_type](#) [tstack](#)  
*Stack of transitions.*
- [hash\\_type](#) [h](#)  
*Map of visited states.*
- const [tgba\\_tba\\_proxy](#) \* [a](#)
- const [state](#) \* [x](#)  
*The state for which we are currently seeking an SCC.*

#### 7.30.1 Detailed Description

Emptiness check on [spot::tgba\\_tba\\_proxy](#) automata using the Magic Search algorithm.

This algorithm comes from

```
@InProceedings{   godefroid.93.pstv,
  author          = {Patrice Godefroid and Gerard .J. Holzmann},
  title           = {On the verification of temporal properties},
  booktitle       = {Proceedings of the 13th IFIP TC6/WG6.1 International
                     Symposium on Protocol Specification, Testing, and
                     Verification (PSTV'93)},
  month           = {May},
  editor          = {Andr{\`e} A. S. Danthine and Guy Leduc
                     and Pierre Wolper},
  address         = {Liege, Belgium},
  pages           = {109--124},
  publisher       = {North-Holland},
  year            = {1993},
  series          = {IFIP Transactions},
  volume          = {C-16},
  isbn            = {0-444-81648-8}
}
```

#### 7.30.2 Member Typedef Documentation

**7.30.2.1** typedef [Sgi::hash\\_map](#)<const [state](#)\*, [magic](#), [state\\_ptr\\_hash](#), [state\\_ptr\\_equal](#)>  
[spot::magic\\_search::hash\\_type](#) [private]

**7.30.2.2** `typedef std::list<state_iter_pair> spot::magic_search::stack_type` [private]

**7.30.2.3** `typedef std::pair<magic_state, tgba_succ_iterator*> spot::magic_search::state_iter_pair` [private]

**7.30.2.4** `typedef std::list<bdd> spot::magic_search::tstack_type` [private]

### 7.30.3 Constructor & Destructor Documentation

**7.30.3.1** `spot::magic_search::magic_search (const tgba_tba_proxy * a)`

Initialize the Magic Search algorithm on the automaton *a*.

**7.30.3.2** `spot::magic_search::~~magic_search ()`

### 7.30.4 Member Function Documentation

**7.30.4.1** `bool spot::magic_search::check ()`

Perform a Magic Search.

#### Returns:

true iff the algorithm has found a new accepting path.

`check()` can be called several times until it return false, to enumerate all accepting paths.

**7.30.4.2** `bool spot::magic_search::has (const state * s, bool m) const` [private]

Check whether we already visited *s* with the Magic bit set to *m*.

**7.30.4.3** `std::ostream& spot::magic_search::print_result (std::ostream & os, const tgba * restrict = 0) const`

Print the last accepting path found.

Restrict printed states to *the* state space of restrict if supplied.

**7.30.4.4** `void spot::magic_search::push (const state * s, bool m)` [private]

Append a new state to the current path.

### 7.30.5 Member Data Documentation

**7.30.5.1** `const tgba_tba_proxy* spot::magic_search::a` [private]

The automata to check.

**7.30.5.2** `hash_type spot::magic_search::h` [private]

Map of visited states.

#### 7.30.5.3 `stack_type spot::magic_search::stack` [private]

Stack of visited states on the path.

#### 7.30.5.4 `tstack_type spot::magic_search::tstack` [private]

Stack of transitions.

This is an addition to the data from the paper.

#### 7.30.5.5 `const state* spot::magic_search::x` [private]

The state for which we are currently seeking an SCC.

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

- [tgbaalgos/magic.hh](#)

### 7.31 `spot::magic_search::magic` Struct Reference

Records whether a state has be seen with the magic bit on or off.

#### Public Attributes

- bool `seen_without`: 1
- bool `seen_with`: 1

#### 7.31.1 Detailed Description

Records whether a state has be seen with the magic bit on or off.

#### 7.31.2 Member Data Documentation

##### 7.31.2.1 bool `spot::magic_search::magic::seen_with`

##### 7.31.2.2 bool `spot::magic_search::magic::seen_without`

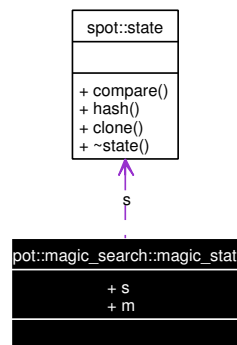
The documentation for this struct was generated from the following file:

- [tgbaalgos/magic.hh](#)

### 7.32 `spot::magic_search::magic_state` Struct Reference

A state for the `spot::magic_search` algorithm.

Collaboration diagram for `spot::magic_search::magic_state`:



### Public Attributes

- const [state](#) \* [s](#)
- bool [m](#)

*The state of the magic demon.*

#### 7.32.1 Detailed Description

A state for the [spot::magic\\_search](#) algorithm.

#### 7.32.2 Member Data Documentation

##### 7.32.2.1 bool [spot::magic\\_search::magic\\_state::m](#)

The state of the magic demon.

##### 7.32.2.2 const [state](#)\* [spot::magic\\_search::magic\\_state::s](#)

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

- [tgbaalgos/magic.hh](#)

## 7.33 spot::minato\_isop Class Reference

Generate an irredundant sum-of-products (ISOP) form of a BDD function.

```
#include <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_`

### 7.33.1 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}
}
```

### 7.33.2 Constructor & Destructor Documentation

#### 7.33.2.1 spot::minato\_isop::minato\_isop (bdd *input*)

Constructor.

- *input* The BDD function to translate in ISOP.

#### 7.33.2.2 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*.

### 7.33.3 Member Function Documentation

#### 7.33.3.1 bdd spot::minato\_isop::next ()

Compute the next sum term of the ISOP form. Return `bddfalse` when all terms have been output.



### 7.33.4 Member Data Documentation

7.33.4.1 std::stack<bdd> [spot::minato\\_isop::cube\\_](#) [private]

7.33.4.2 bdd [spot::minato\\_isop::ret\\_](#) [private]

7.33.4.3 std::stack<[local\\_vars](#)> [spot::minato\\_isop::todo\\_](#) [private]

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

- misc/[minato.hh](#)

## 7.34 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](#)

### 7.34.1 Detailed Description

Internal variables for [minato\\_isop](#).

### 7.34.2 Member Enumeration Documentation

#### 7.34.2.1 anonymous enum

Enumeration values:

*FirstStep*

*SecondStep**ThirdStep**FourthStep*

### 7.34.3 Constructor & Destructor Documentation

7.34.3.1 `spot::minato_isop::local_vars::local_vars (bdd f_min, bdd f_max, bdd vars)` `[inline]`

### 7.34.4 Member Data Documentation

7.34.4.1 `bdd spot::minato_isop::local_vars::f0_max`

7.34.4.2 `bdd spot::minato_isop::local_vars::f0_min`

7.34.4.3 `bdd spot::minato_isop::local_vars::f1_max`

7.34.4.4 `bdd spot::minato_isop::local_vars::f1_min`

7.34.4.5 `bdd spot::minato_isop::local_vars::f_max`

7.34.4.6 `bdd spot::minato_isop::local_vars::f_min`

7.34.4.7 `bdd spot::minato_isop::local_vars::g0`

7.34.4.8 `bdd spot::minato_isop::local_vars::g1`

7.34.4.9 `enum { ... } spot::minato_isop::local_vars::step`

7.34.4.10 `bdd spot::minato_isop::local_vars::v1`

7.34.4.11 `bdd spot::minato_isop::local_vars::vars`

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

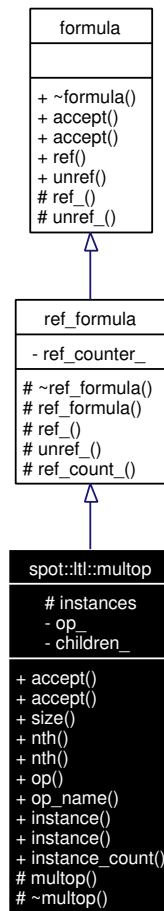
- [misc/minato.hh](#)

## 7.35 spot::ltl::multop Class Reference

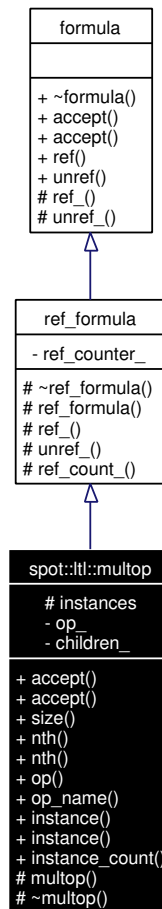
Multi-operand operators.

```
#include <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 *n*th 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*

### Static Public Member Functions

- `formula * instance` (type op, `formula *first`, `formula *second`)  
*Build a spot::ltl::multop with two children.*
- `formula * instance` (type op, `vec *v`)  
*Build a spot::ltl::multop with many children.*
- `unsigned instance_count` ()  
*Number of instantiated multi-operand operators. For debugging.*
- `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.*

### Static Protected Attributes

- [map instances](#)

### Private Attributes

- [type op\\_](#)
- [vec \\* children\\_](#)

### 7.35.1 Detailed Description

Multi-operand operators.

These operators are considered commutative and associative.

### 7.35.2 Member Typedef Documentation

**7.35.2.1** `typedef std::map<pair, formula\*, pairemp> spot::ltl::multop::map [protected]`

**7.35.2.2** `typedef std::pair<type, vec\*> spot::ltl::multop::pair [protected]`

**7.35.2.3** `typedef std::vector<formula\*> spot::ltl::multop::vec`

List of formulae.

### 7.35.3 Member Enumeration Documentation

**7.35.3.1** `enum spot::ltl::multop::type`

Enumeration values:

*Or*

*And*

### 7.35.4 Constructor & Destructor Documentation

**7.35.4.1** `spot::ltl::multop::multop (type op, vec \* v) [protected]`

**7.35.4.2** `virtual spot::ltl::multop::~~multop () [protected, virtual]`

### 7.35.5 Member Function Documentation

**7.35.5.1** `virtual void spot::ltl::multop::accept (const\_visitor & v) const [virtual]`

Entry point for `vspot::ltl::const_visitor` instances.

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

**7.35.5.2 virtual void spot::ltl::multop::accept (visitor & v) [virtual]**

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

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

**7.35.5.3 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.

**7.35.5.4 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.

**7.35.5.5 unsigned spot::ltl::multop::instance\_count () [static]**

Number of instantiated multi-operand operators. For debugging.

**7.35.5.6 formula\* spot::ltl::multop::nth (unsigned n)**

Get the nth children.

Starting with  $n = 0$ .

**7.35.5.7 const formula\* spot::ltl::multop::nth (unsigned n) const**

Get the nth children.

Starting with  $n = 0$ .

**7.35.5.8 type spot::ltl::multop::op () const**

Get the type of this operator.

**7.35.5.9 const char\* spot::ltl::multop::op\_name () const**

Get the type of this operator, as a string.

**7.35.5.10 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.

#### 7.35.5.11 `void spot::ltl::ref_formula::ref_()` [protected, virtual, inherited]

increment reference counter if any

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

#### 7.35.5.12 `unsigned spot::ltl::ref_formula::ref_count_()` [protected, inherited]

Number of references to this formula.

#### 7.35.5.13 `unsigned spot::ltl::multop::size() const`

Get the number of children.

#### 7.35.5.14 `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.

#### 7.35.5.15 `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`.

### 7.35.6 Member Data Documentation

#### 7.35.6.1 `vec* spot::ltl::multop::children_` [private]

#### 7.35.6.2 `map spot::ltl::multop::instances` [static, protected]

#### 7.35.6.3 `type spot::ltl::multop::op_` [private]

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

- `ltlast/multop.hh`

## 7.36 `spot::ltl::multop::pairecmp` Struct Reference

Comparison functor used internally by `ltl::multop`.

```
#include <multop.hh>
```



### Public Member Functions

- bool [operator\(\)](#) (const [pair](#) &p1, const [pair](#) &p2) const

#### 7.36.1 Detailed Description

Comparison functor used internally by [ltl::multop](#).

#### 7.36.2 Member Function Documentation

**7.36.2.1** 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:

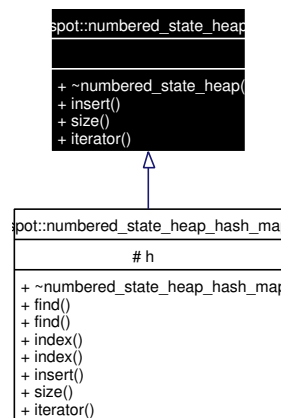
- [ltlast/multop.hh](#)

## 7.37 spot::numbered\_state\_heap Class Reference

Keep track of a large quantity of indexed states.

```
#include <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 [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

### 7.37.1 Detailed Description

Keep track of a large quantity of indexed states.

### 7.37.2 Member Typedef Documentation

**7.37.2.1** `typedef std::pair<const state*, int> spot::numbered\_state\_heap::state\_index`

**7.37.2.2** `typedef std::pair<const state*, int*> spot::numbered\_state\_heap::state\_index\_p`

### 7.37.3 Constructor & Destructor Documentation

**7.37.3.1** `virtual spot::numbered\_state\_heap::~numbered\_state\_heap () [inline, virtual]`

### 7.37.4 Member Function Documentation

**7.37.4.1** `virtual state\_index\_p spot::numbered\_state\_heap::find (const state * s) [pure virtual]`

Implemented in [spot::numbered\\_state\\_heap\\_hash\\_map](#).

**7.37.4.2** `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* in the heap with index. *s* will be freed if it is different of *p*.

There 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](#).

**7.37.4.3** `virtual state\_index\_p spot::numbered_state_heap::index (const state * s) [pure virtual]`

Implemented in [spot::numbered\\_state\\_heap\\_hash\\_map](#).

**7.37.4.4** `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](#).

**7.37.4.5** `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](#).

**7.37.4.6** `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](#).

**7.37.4.7** `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:

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

## 7.38 `spot::numbered_state_heap_const_iterator` Class Reference

Iterator on [numbered\\_state\\_heap](#) objects.

```
#include <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

### 7.38.1 Detailed Description

Iterator on [numbered\\_state\\_heap](#) objects.

### 7.38.2 Constructor & Destructor Documentation

**7.38.2.1** virtual [spot::numbered\\_state\\_heap\\_const\\_iterator::~~numbered\\_state\\_heap\\_const\\_iterator](#) () [inline, virtual]

### 7.38.3 Member Function Documentation

**7.38.3.1** virtual bool [spot::numbered\\_state\\_heap\\_const\\_iterator::done](#) () const [pure virtual]

**7.38.3.2** virtual void [spot::numbered\\_state\\_heap\\_const\\_iterator::first](#) () [pure virtual]

Iteration.

**7.38.3.3** virtual int [spot::numbered\\_state\\_heap\\_const\\_iterator::get\\_index](#) () const [pure virtual]

**7.38.3.4** virtual const [state](#)\* [spot::numbered\\_state\\_heap\\_const\\_iterator::get\\_state](#) () const [pure virtual]

Inspection.

**7.38.3.5** virtual void [spot::numbered\\_state\\_heap\\_const\\_iterator::next](#) () [pure virtual]

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

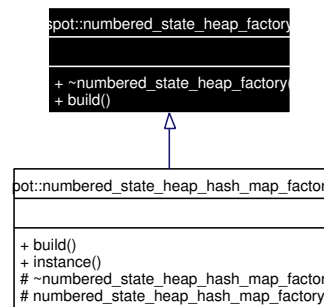
- [tgbaalgos/gtec/nsheap.hh](#)

## 7.39 spot::numbered\_state\_heap\_factory Class Reference

Abstract factory for [numbered\\_state\\_heap](#).

```
#include <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

#### 7.39.1 Detailed Description

Abstract factory for `numbered_state_heap`.

#### 7.39.2 Constructor & Destructor Documentation

**7.39.2.1** virtual `spot::numbered_state_heap_factory::~~numbered_state_heap_factory` ()  
 [inline, virtual]

#### 7.39.3 Member Function Documentation

**7.39.3.1** 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:

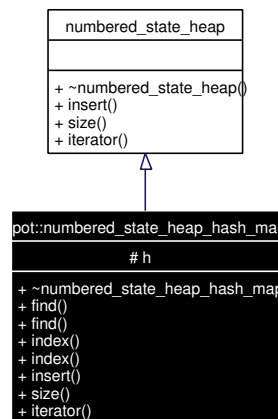
- `tgbaalgos/gtec/nsheap.hh`

## 7.40 spot::numbered\_state\_heap\_hash\_map Class Reference

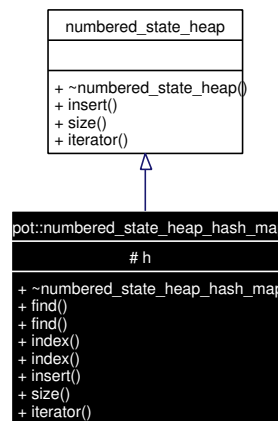
A straightforward implementation of `numbered_state_heap` with a hash map.

```
#include <nsheap.hh>
```

Inheritance diagram for `spot::numbered_state_heap_hash_map`:



Collaboration diagram for `spot::numbered_state_heap_hash_map`:



## 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\\_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 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 Types

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

### Protected Attributes

- [hash\\_type](#) *h*  
Map of visited states.

## 7.40.1 Detailed Description

A straightforward implementation of [numbered\\_state\\_heap](#) with a hash map.

## 7.40.2 Member Typedef Documentation

**7.40.2.1** typedef Sgi::hash\_map<const [state](#)\*, int, [state\\_ptr\\_hash](#), [state\\_ptr\\_equal](#)>  
[spot::numbered\\_state\\_heap\\_hash\\_map::hash\\_type](#) [protected]

**7.40.2.2** typedef std::pair<const [state](#)\*, int> [spot::numbered\\_state\\_heap::state\\_index](#)  
[inherited]

**7.40.2.3** typedef std::pair<const [state](#)\*, int\*> [spot::numbered\\_state\\_heap::state\\_index\\_p](#)  
[inherited]

## 7.40.3 Constructor & Destructor Documentation

**7.40.3.1** virtual [spot::numbered\\_state\\_heap\\_hash\\_map::~~numbered\\_state\\_heap\\_hash\\_map](#) ()  
[virtual]

## 7.40.4 Member Function Documentation

**7.40.4.1** virtual [state\\_index\\_p](#) [spot::numbered\\_state\\_heap\\_hash\\_map::find](#) (const [state](#) \* *s*)  
[virtual]

Implements [spot::numbered\\_state\\_heap](#).

**7.40.4.2** `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. *s* will be freed if it is different of *p*.

There 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`.

**7.40.4.3** `virtual state_index_p spot::numbered_state_heap_hash_map::index (const state * s)` [virtual]

Implements `spot::numbered_state_heap`.

**7.40.4.4** `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`.

**7.40.4.5** `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`.

**7.40.4.6** `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`.

**7.40.4.7** `virtual int spot::numbered_state_heap_hash_map::size () const` [virtual]

The number of stored states.

Implements `spot::numbered_state_heap`.

## 7.40.5 Member Data Documentation

**7.40.5.1** `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:



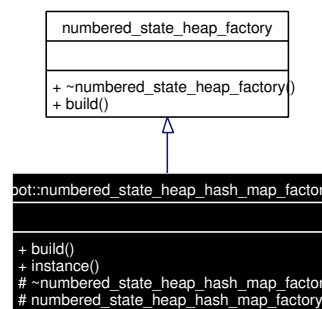
- [tgbaalgos/gtec/nsheap.hh](#)

## 7.41 spot::numbered\_state\_heap\_hash\_map\_factory Class Reference

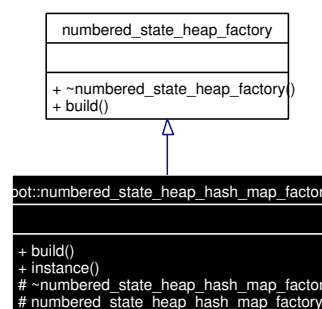
Factory for [numbered\\_state\\_heap\\_hash\\_map](#).

```
#include <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

- 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` ()

### 7.41.1 Detailed Description

Factory for [numbered\\_state\\_heap\\_hash\\_map](#).

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

### 7.41.2 Constructor & Destructor Documentation

**7.41.2.1** virtual [spot::numbered\\_state\\_heap\\_hash\\_map\\_factory::~~numbered\\_state\\_heap\\_hash\\_map\\_factory\(\)](#) [inline, protected, virtual]

**7.41.2.2** [spot::numbered\\_state\\_heap\\_hash\\_map\\_factory::numbered\\_state\\_heap\\_hash\\_map\\_factory\(\)](#) [protected]

### 7.41.3 Member Function Documentation

**7.41.3.1** virtual [numbered\\_state\\_heap\\_hash\\_map\\*](#) [spot::numbered\\_state\\_heap\\_hash\\_map\\_factory::build\(\)](#) const [virtual]

Implements [spot::numbered\\_state\\_heap\\_factory](#).

**7.41.3.2** 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:

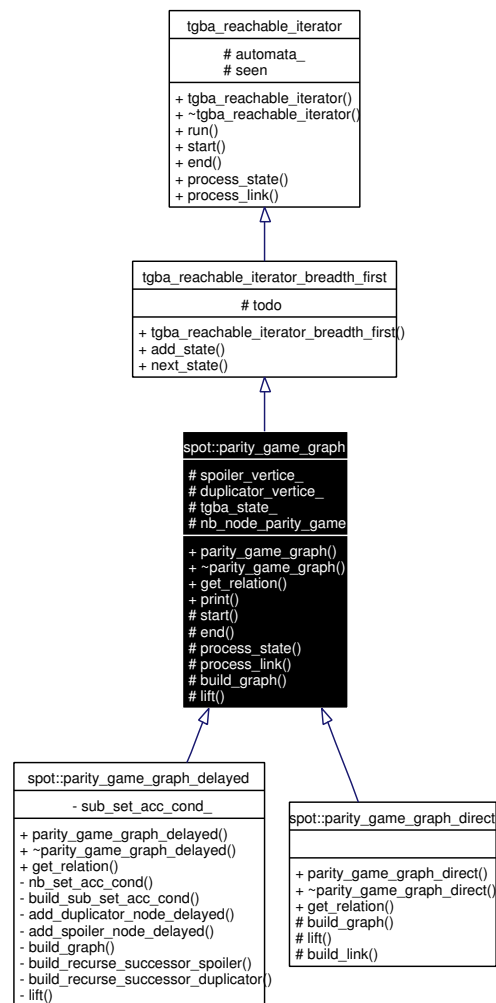
- [tgbaalgos/gtec/nsheap.hh](#)

## 7.42 spot::parity\_game\_graph Class Reference

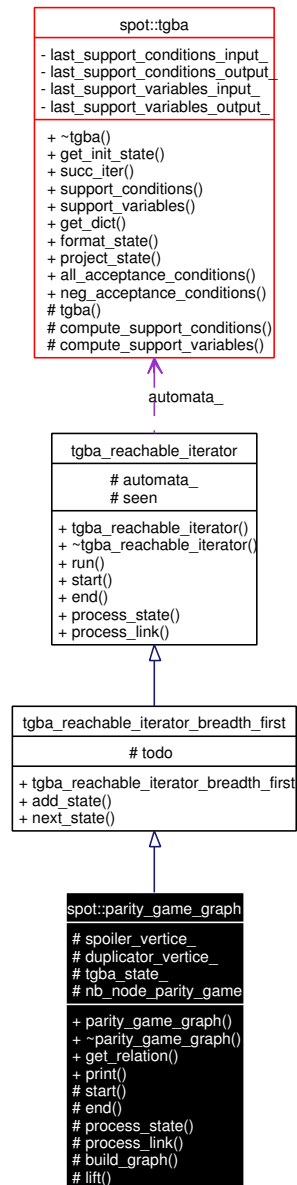
Parity game graph which compute a simulation relation.

```
#include <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](#).

### 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  
*Compute the link of the graph. Successor of spoiler node (resp. duplicator node) are duplicator node (resp. spoiler node). 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.*

#### 7.42.1 Detailed Description

Parity game graph which compute a simulation relation.

#### 7.42.2 Member Typedef Documentation

**7.42.2.1** 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](#).

### 7.42.3 Constructor & Destructor Documentation

**7.42.3.1** `spot::parity_game_graph::parity_game_graph (const tgba * a)`

**7.42.3.2** `virtual spot::parity_game_graph::~parity\_game\_graph ()` [virtual]

### 7.42.4 Member Function Documentation

**7.42.4.1** `virtual void spot::tgba_reachable_iterator_breadth_first::add_state (const state * s)`  
[virtual, inherited]

Implements [spot::tgba\\_reachable\\_iterator](#).

**7.42.4.2** `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](#).

**7.42.4.3** `void spot::parity_game_graph::end ()` [protected, virtual]

Called by [run\(\)](#) once all states have been explored.

Reimplemented from [spot::tgba\\_reachable\\_iterator](#).

**7.42.4.4** `virtual simulation\_relation* spot::parity_game_graph::get_relation ()` [pure virtual]

Implemented in [spot::parity\\_game\\_graph\\_direct](#), and [spot::parity\\_game\\_graph\\_delayed](#).

**7.42.4.5** `virtual void spot::parity_game_graph::lift ()` [protected, pure virtual]

Compute the link of the graph. Successor of spoiler node (resp. duplicator node) are duplicator node (resp. spoiler node). 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](#).

**7.42.4.6** `virtual const state* spot::tgba_reachable_iterator_breadth_first::next_state ()`  
[virtual, inherited]

Called by [run\(\)](#) to obtain the.

Implements [spot::tgba\\_reachable\\_iterator](#).

**7.42.4.7** `void spot::parity_game_graph::print (std::ostream & os)`

**7.42.4.8** `void spot::parity_game_graph::process_link (int in, int out, const tgba\_succ\_iterator * si)`  
[protected, virtual]

Called by [run\(\)](#) to process a transition.

**Parameters:**

- in* The source state number.
- out* The destination state number.
- si* The [spot::tgba\\_succ\\_iterator](#) positionned on the current transition.

Reimplemented from [spot::tgba\\_reachable\\_iterator](#).

**7.42.4.9 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* An unique number assigned to *s*.
- si* The [spot::tgba\\_succ\\_iterator](#) for *s*.

Reimplemented from [spot::tgba\\_reachable\\_iterator](#).

**7.42.4.10 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.

**7.42.4.11 void spot::parity\_game\_graph::start ()** [protected, virtual]

Called by [run\(\)](#) before starting its iteration.

Reimplemented from [spot::tgba\\_reachable\\_iterator](#).

**7.42.5 Member Data Documentation**

**7.42.5.1 const [tgba](#)\* spot::tgba\_reachable\_iterator::automata\_** [protected, inherited]

The [spot::tgba](#) to explore.

**7.42.5.2 [dn\\_v](#) spot::parity\_game\_graph::duplicator\_vertice\_** [protected]

**7.42.5.3 int spot::parity\_game\_graph::nb\_node\_parity\_game** [protected]

**7.42.5.4 [seen\\_map](#) spot::tgba\_reachable\_iterator::seen** [protected, inherited]

States already seen.

**7.42.5.5 [sn\\_v](#) spot::parity\_game\_graph::spoiler\_vertice\_** [protected]

**7.42.5.6 [s\\_v](#) spot::parity\_game\_graph::tgba\_state\_** [protected]

#### 7.42.5.7 std::deque<const state\*> spot::tgba\_reachable\_iterator\_breadth\_first::todo [protected, inherited]

A queue of states yet to explore.

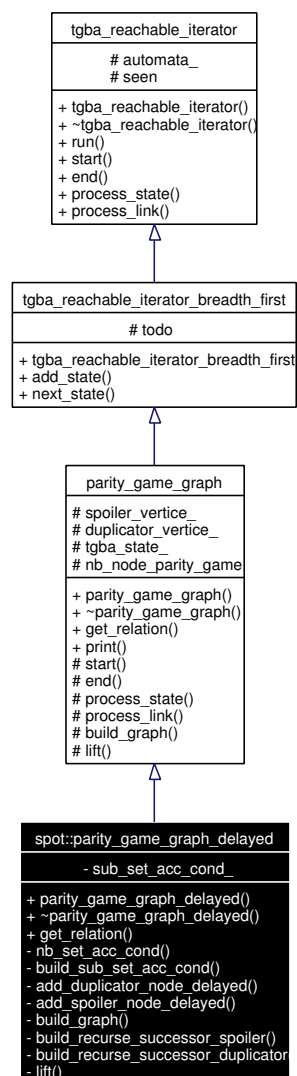
The documentation for this class was generated from the following file:

- tgbaalgs/reductgba\_sim.hh

### 7.43 spot::parity\_game\_graph\_delayed Class Reference

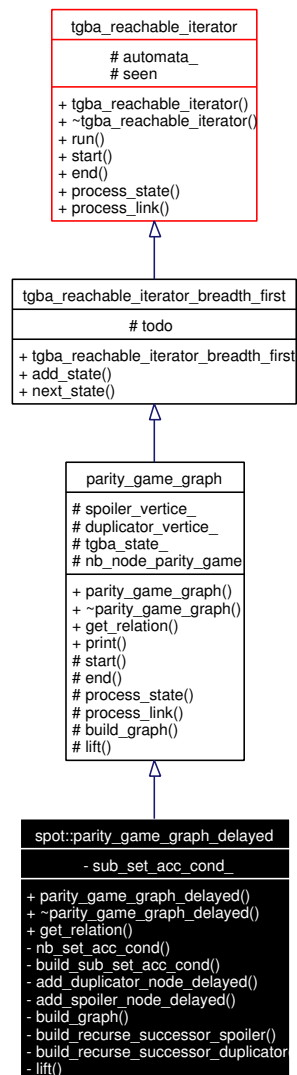
```
#include <reductgba_sim.hh>
```

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 `simulation_relation` \* `get_relation` ()
- 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`.*

### 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.*
- void [build\\_sub\\_set\\_acc\\_cond](#) ()  
*Compute [sub\\_set\\_acc\\_cond\\_](#).*
- [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)
- virtual void [build\\_graph](#) ()  
*Compute the couple as for direct simulation.,.*

- 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 [lift](#) ()

*The Jurdzinski's lifting algorithm.*

### Private Attributes

- [bdd\\_v sub\\_set\\_acc\\_cond\\_](#)

#### 7.43.1 Detailed Description

Parity game graph which compute the delayed simulation relation as explain in { icalp2001, AUTHOR = {Etessami, Thomas Wilke, Rebecca A. Schuller}, TITLE = {Fair Simulation Relations, Parity Games, and State Space Reduction for Buchi Automata}, BOOKTITLE = {Automata, Languages and Programming, 28th international colloquium}, PAGES = {694–707}, YEAR = 2001, EDITOR = {Orejas, Fernando and Spirakis, Paul G. and van Leeuwen, Jan}, VOLUME = 2076, SERIES = {Lecture Notes in Computer Science}, ADDRESS = {Crete, Greece}, MONTH = JUL, PUBLISHER = {Springer}, url = {cite-seer.ist.psu.edu/472661.html} }

#### 7.43.2 Member Typedef Documentation

**7.43.2.1** `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.

**7.43.2.2** `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](#).

#### 7.43.3 Constructor & Destructor Documentation

**7.43.3.1** `spot::parity_game_graph_delayed::parity_game_graph_delayed (const tgba * a)`

**7.43.3.2** `spot::parity_game_graph_delayed::~~parity_game_graph_delayed ()`

#### 7.43.4 Member Function Documentation

**7.43.4.1** `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]`

**7.43.4.2** `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]`

**7.43.4.3** `virtual void spot::tgba_reachable_iterator_breadth_first::add_state (const state * s)` [virtual, inherited]

Implements [spot::tgba\\_reachable\\_iterator](#).

**7.43.4.4** `virtual void spot::parity_game_graph_delayed::build_graph ()` [private, virtual]

Compute the couple as for direct simulation,.

Implements [spot::parity\\_game\\_graph](#).

**7.43.4.5** `void spot::parity_game_graph_delayed::build_recurse_successor_duplicator (duplicator\_node * dn, spoiler\_node * sn, std::ostream & os)` [private]

**7.43.4.6** `void spot::parity_game_graph_delayed::build_recurse_successor_spoiler (spoiler\_node * sn, std::ostream & os)` [private]

**7.43.4.7** `void spot::parity_game_graph_delayed::build_sub_set_acc_cond ()` [private]

Compute sub\_set\_acc\_cond;.

**7.43.4.8** `void spot::parity_game_graph::end ()` [protected, virtual, inherited]

Called by [run\(\)](#) once all states have been explored.

Reimplemented from [spot::tgba\\_reachable\\_iterator](#).

**7.43.4.9** `virtual simulation\_relation* spot::parity_game_graph_delayed::get_relation ()` [virtual]

Implements [spot::parity\\_game\\_graph](#).

**7.43.4.10** `virtual void spot::parity_game_graph_delayed::lift ()` [private, virtual]

The Jurdzinski's lifting algorithm.

Implements [spot::parity\\_game\\_graph](#).

**7.43.4.11** `int spot::parity_game_graph_delayed::nb_set_acc_cond ()` [private]

Return the number of acceptance condition.

**7.43.4.12** `virtual const state* spot::tgba_reachable_iterator_breadth_first::next_state ()` [virtual, inherited]

Called by [run\(\)](#) to obtain the.

Implements [spot::tgba\\_reachable\\_iterator](#).

**7.43.4.13** `void spot::parity_game_graph::print (std::ostream & os)` [inherited]

**7.43.4.14** `void spot::parity_game_graph::process_link (int in, int out, const tgba_succ_iterator * si)` [protected, virtual, inherited]

Called by `run()` to process a transition.

**Parameters:**

*in* The source state number.

*out* The destination state number.

*si* The `spot::tgba_succ_iterator` positionned on the current transition.

Reimplemented from `spot::tgba_reachable_iterator`.

**7.43.4.15** `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* An unique number assigned to *s*.

*si* The `spot::tgba_succ_iterator` for *s*.

Reimplemented from `spot::tgba_reachable_iterator`.

**7.43.4.16** `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.

**7.43.4.17** `void spot::parity_game_graph::start ()` [protected, virtual, inherited]

Called by `run()` before starting its iteration.

Reimplemented from `spot::tgba_reachable_iterator`.

## 7.43.5 Member Data Documentation

**7.43.5.1** `const tgba* spot::tgba_reachable_iterator::automata_` [protected, inherited]

The `spot::tgba` to explore.

**7.43.5.2** `dn_v spot::parity_game_graph::duplicator_vertice_` [protected, inherited]

**7.43.5.3** `int spot::parity_game_graph::nb_node_parity_game` [protected, inherited]

**7.43.5.4** `seen_map spot::tgba_reachable_iterator::seen` [protected, inherited]

States already seen.

7.43.5.5 [sn\\_v spot::parity\\_game\\_graph::spoiler\\_vertice\\_](#) [protected, inherited]

7.43.5.6 [bdd\\_v spot::parity\\_game\\_graph\\_delayed::sub\\_set\\_acc\\_cond\\_](#) [private]

7.43.5.7 [s\\_v spot::parity\\_game\\_graph::tgba\\_state\\_](#) [protected, inherited]

7.43.5.8 [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:

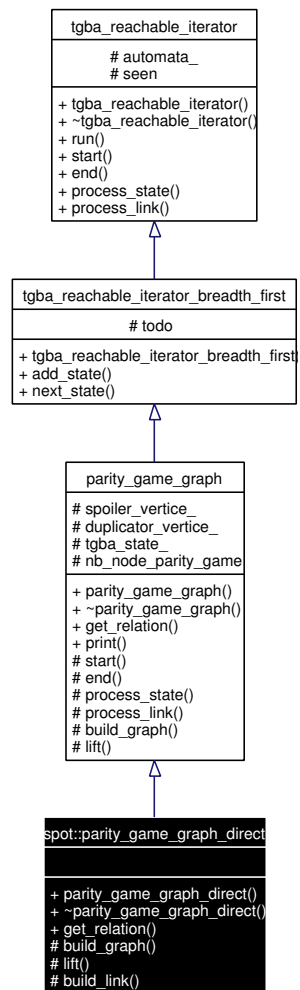
- [tgbaalgos/reductgba\\_sim.hh](#)

## 7.44 spot::parity\_game\_graph\_direct Class Reference

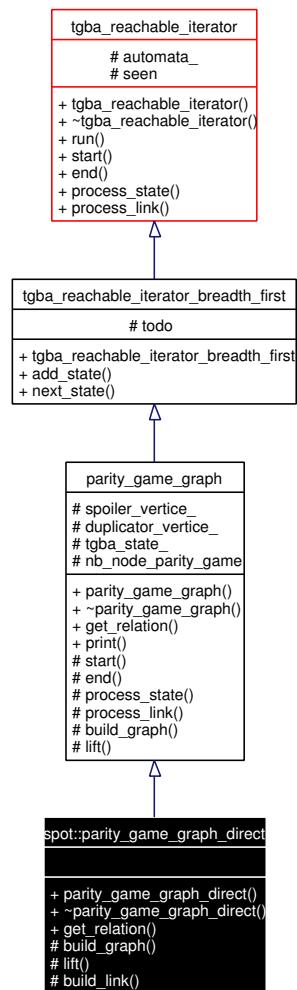
Parity game graph which compute the direct simulation relation.

```
#include <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 [simulation\\_relation](#) \* [get\\_relation](#) ()
- 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](#).*

## 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](#) ()  
*Compute the link of the graph. Successor of spoiler node (resp. duplicator node) are duplicator node (resp. spoiler node). 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.*

**7.44.1 Detailed Description**

Parity game graph which compute the direct simulation relation.

**7.44.2 Member Typedef Documentation**

**7.44.2.1** 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](#).

**7.44.3 Constructor & Destructor Documentation**

**7.44.3.1** [spot::parity\\_game\\_graph\\_direct::parity\\_game\\_graph\\_direct](#) (const [tgba](#) \* a)

## 7.44.3.2 spot::parity\_game\_graph\_direct::~~parity\_game\_graph\_direct ()

## 7.44.4 Member Function Documentation

## 7.44.4.1 virtual void spot::tgba\_reachable\_iterator\_breadth\_first::add\_state (const state \* s) [virtual, inherited]

Implements [spot::tgba\\_reachable\\_iterator](#).

## 7.44.4.2 virtual void spot::parity\_game\_graph\_direct::build\_graph () [protected, virtual]

Compute each node of the graph.

Implements [spot::parity\\_game\\_graph](#).

## 7.44.4.3 void spot::parity\_game\_graph\_direct::build\_link () [protected]

## 7.44.4.4 void spot::parity\_game\_graph::end () [protected, virtual, inherited]

Called by [run\(\)](#) once all states have been explored.

Reimplemented from [spot::tgba\\_reachable\\_iterator](#).

## 7.44.4.5 virtual simulation\_relation\* spot::parity\_game\_graph\_direct::get\_relation () [virtual]

Implements [spot::parity\\_game\\_graph](#).

## 7.44.4.6 virtual void spot::parity\_game\_graph\_direct::lift () [protected, virtual]

Compute the link of the graph. Successor of spoiler node (resp. duplicator node) are duplicator node (resp. spoiler node). 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](#).

## 7.44.4.7 virtual const state\* spot::tgba\_reachable\_iterator\_breadth\_first::next\_state () [virtual, inherited]

Called by [run\(\)](#) to obtain the.

Implements [spot::tgba\\_reachable\\_iterator](#).

## 7.44.4.8 void spot::parity\_game\_graph::print (std::ostream &amp; os) [inherited]

## 7.44.4.9 void spot::parity\_game\_graph::process\_link (int in, int out, const tgba\_succ\_iterator \* si) [protected, virtual, inherited]

Called by [run\(\)](#) to process a transition.

**Parameters:**

*in* The source state number.

*out* The destination state number.

*si* The [spot::tgba\\_succ\\_iterator](#) positionned on the current transition.

Reimplemented from [spot::tgba\\_reachable\\_iterator](#).

**7.44.4.10** `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* An unique number assigned to *s*.

*si* The [spot::tgba\\_succ\\_iterator](#) for *s*.

Reimplemented from [spot::tgba\\_reachable\\_iterator](#).

**7.44.4.11** `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.

**7.44.4.12** `void spot::parity_game_graph::start ()` [protected, virtual, inherited]

Called by [run\(\)](#) before starting its iteration.

Reimplemented from [spot::tgba\\_reachable\\_iterator](#).

### 7.44.5 Member Data Documentation

**7.44.5.1** `const tgba* spot::tgba_reachable_iterator::automata_` [protected, inherited]

The [spot::tgba](#) to explore.

**7.44.5.2** `dn_v spot::parity_game_graph::duplicator_vertice_` [protected, inherited]

**7.44.5.3** `int spot::parity_game_graph::nb_node_parity_game` [protected, inherited]

**7.44.5.4** `seen_map spot::tgba_reachable_iterator::seen` [protected, inherited]

States already seen.

**7.44.5.5** `sn_v spot::parity_game_graph::spoiler_vertice_` [protected, inherited]

**7.44.5.6** `s_v spot::parity_game_graph::tgba_state_` [protected, inherited]

#### 7.44.5.7 `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`

## 7.45 yy::Position Class Reference

Abstract a `Position`.

```
#include <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

- `const unsigned int initial_column = 0`  
*Initial column number.*
- `const unsigned int initial_line = 1`  
*Initial line number.*

### 7.45.1 Detailed Description

Abstract a [Position](#).

### 7.45.2 Constructor & Destructor Documentation

#### 7.45.2.1 yy::Position::Position () [inline]

Construct a [Position](#).

### 7.45.3 Member Function Documentation

#### 7.45.3.1 void yy::Position::columns (int *count* = 1) [inline]

(column related) Advance to the COUNT next columns.

#### 7.45.3.2 void yy::Position::lines (int *count* = 1) [inline]

(line related) Advance to the COUNT next lines.

### 7.45.4 Member Data Documentation

#### 7.45.4.1 unsigned int yy::Position::column

Current column number.

#### 7.45.4.2 std::string yy::Position::filename

File name to which this position refers.

#### 7.45.4.3 const unsigned int yy::Position::initial\_column = 0 [static]

Initial column number.

#### 7.45.4.4 const unsigned int yy::Position::initial\_line = 1 [static]

Initial line number.

#### 7.45.4.5 unsigned int yy::Position::line

Current line number.

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

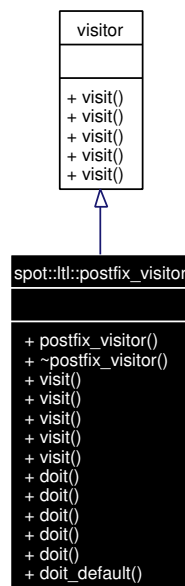
- [ltlparse/position.hh](#)

## 7.46 spot::ltl::postfix\_visitor Class Reference

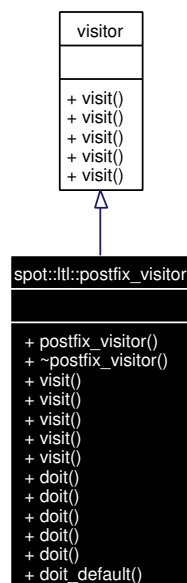
Apply an algorithm on each node of an AST, during a postfix traversal.

```
#include <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)

### 7.46.1 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.

### 7.46.2 Constructor & Destructor Documentation

**7.46.2.1** [spot::ltl::postfix\\_visitor::postfix\\_visitor \(\)](#)

**7.46.2.2** [virtual spot::ltl::postfix\\_visitor::~~postfix\\_visitor \(\)](#) [virtual]

### 7.46.3 Member Function Documentation

**7.46.3.1** [virtual void spot::ltl::postfix\\_visitor::doit \(\[constant\]\(#\) \\* c\)](#) [virtual]

**7.46.3.2** [virtual void spot::ltl::postfix\\_visitor::doit \(\[multop\]\(#\) \\* mo\)](#) [virtual]

**7.46.3.3** [virtual void spot::ltl::postfix\\_visitor::doit \(\[binop\]\(#\) \\* bo\)](#) [virtual]

**7.46.3.4** [virtual void spot::ltl::postfix\\_visitor::doit \(\[unop\]\(#\) \\* uo\)](#) [virtual]

**7.46.3.5** [virtual void spot::ltl::postfix\\_visitor::doit \(\[atomic\\\_prop\]\(#\) \\* ap\)](#) [virtual]

**7.46.3.6** [virtual void spot::ltl::postfix\\_visitor::doit\\_default \(\[formula\]\(#\) \\* f\)](#) [virtual]

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

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

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

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

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

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

**7.46.3.10** `void spot::ltl::postfix_visitor::visit (unop *uo) [virtual]`

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

**7.46.3.11** `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](#)

## 7.47 `spot::ptr_hash< T >` Struct Template Reference

A hash function for pointers.

```
#include <hash.hh>
```

### Public Member Functions

- `size_t operator() (const T *p) const`

#### 7.47.1 Detailed Description

`template<class T> struct spot::ptr_hash< T >`

A hash function for pointers.

#### 7.47.2 Member Function Documentation

**7.47.2.1** `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](#)

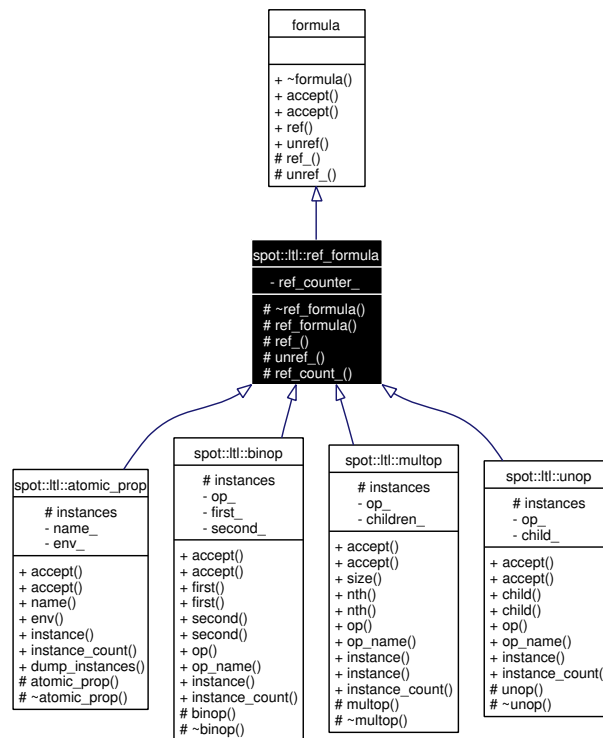
## 7.48 `spot::ltl::ref_formula` Class Reference

A reference-counted LTL formula.

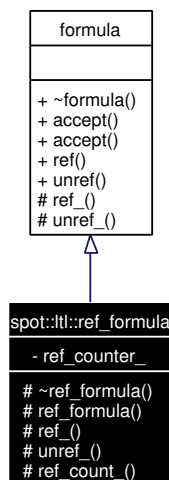
```
#include <refformula.hh>
```

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*

### Static Public Member Functions

- `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.*

### Private Attributes

- `unsigned ref_counter_`

## 7.48.1 Detailed Description

A reference-counted LTL formula.

## 7.48.2 Constructor & Destructor Documentation

**7.48.2.1** `virtual spot::ltl::ref_formula::~~ref_formula ()` [protected, virtual]

**7.48.2.2** `spot::ltl::ref_formula::ref_formula ()` [protected]

## 7.48.3 Member Function Documentation

**7.48.3.1** `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`.

**7.48.3.2 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](#).

**7.48.3.3 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.

**7.48.3.4 void spot::ltl::ref\_formula::ref () [protected, virtual]**

increment reference counter if any

Reimplemented from [spot::ltl::formula](#).

**7.48.3.5 unsigned spot::ltl::ref\_formula::ref\_count\_ () [protected]**

Number of references to this formula.

**7.48.3.6 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.

**7.48.3.7 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](#).

**7.48.4 Member Data Documentation****7.48.4.1 unsigned spot::ltl::ref\_formula::ref\_counter\_ [private]**

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

- Itlast/[reformula.hh](#)

**7.49 spot::scc\_stack Class Reference**

```
#include <sccstack.hh>
```

**Public Types**

- typedef std::stack< [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.*
- void [pop](#) ()  
*Pop the top SCC.*
- size\_t [size](#) () const  
*How many SCC are in stack.*
- bool [empty](#) () const  
*Is the stack empty?*

### Public Attributes

- [stack\\_type](#) s

#### 7.49.1 Member Typedef Documentation

**7.49.1.1** `typedef std::stack<connected\_component> spot::scc\_stack::stack\_type`

#### 7.49.2 Member Function Documentation

**7.49.2.1** `bool spot::scc_stack::empty () const`

Is the stack empty?

**7.49.2.2** `void spot::scc_stack::pop ()`

Pop the top SCC.

**7.49.2.3** `void spot::scc_stack::push (int index)`

Stack a new SCC with index *index*.

**7.49.2.4** `size_t spot::scc_stack::size () const`

How many SCC are in stack.

**7.49.2.5** `connected\_component& spot::scc_stack::top ()`

Access the top SCC.

### 7.49.3 Member Data Documentation

#### 7.49.3.1 [stack\\_type spot::scc\\_stack::s](#)

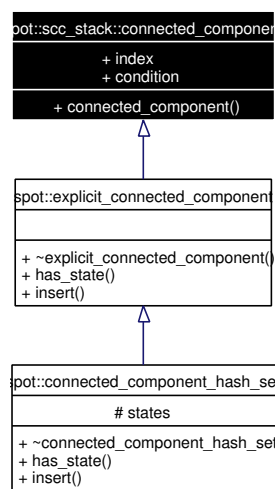
The documentation for this class was generated from the following file:

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

## 7.50 spot::scc\_stack::connected\_component Struct Reference

```
#include <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](#)

### 7.50.1 Constructor & Destructor Documentation

#### 7.50.1.1 `spot::scc_stack::connected_component::connected_component` (int *index* = -1)

### 7.50.2 Member Data Documentation

#### 7.50.2.1 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.

### 7.50.2.2 int spot::scc\_stack::connected\_component::index

Index of the SCC.

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

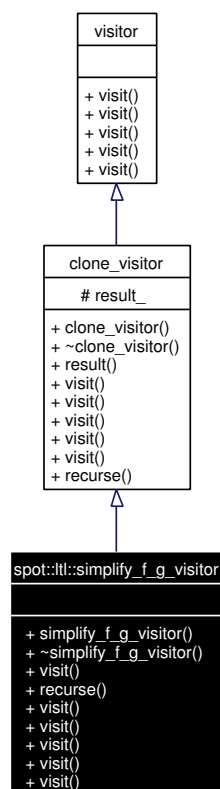
- [tgbaalgorithms/gtec/sccstack.hh](#)

## 7.51 spot::ltl::simplify\_f\_g\_visitor Class Reference

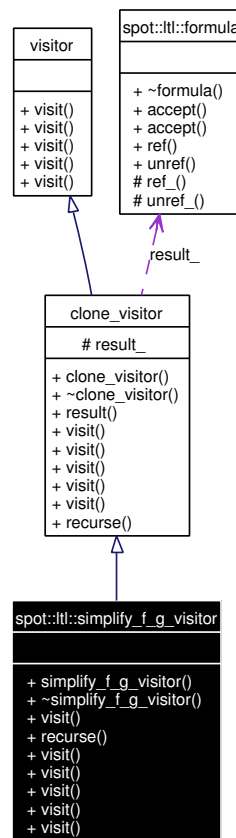
Replace `true U f` and `false R g` by `F f` and `G g`.

```
#include <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)`
- `void visit (atomic_prop *ap)`
- `void visit (unop *uo)`
- `void visit (binop *bo)`
- `void visit (multop *mo)`
- `void visit (constant *c)`
- `formula * result () const`

### Protected Attributes

- `formula * result_`

### Private Types

- `typedef clone_visitor super`

### 7.51.1 Detailed Description

Replace `true`  $\cup$  `f` and `false`  $\cap$  `g` by `F` `f` and `G` `g`.

### 7.51.2 Member Typedef Documentation

7.51.2.1 `typedef` [clone\\_visitor](#) `spot::ltl::simplify_f_g_visitor::super` `[private]`

### 7.51.3 Constructor & Destructor Documentation

7.51.3.1 `spot::ltl::simplify_f_g_visitor::simplify_f_g_visitor()`

7.51.3.2 `virtual` `spot::ltl::simplify_f_g_visitor::~~simplify_f_g_visitor()` `[virtual]`

### 7.51.4 Member Function Documentation

7.51.4.1 `virtual` [formula](#)\* `spot::ltl::simplify_f_g_visitor::recurse` ([formula](#) \* `f`) `[virtual]`

Reimplemented from [spot::ltl::clone\\_visitor](#).

7.51.4.2 [formula](#)\* `spot::ltl::clone_visitor::result()` `const` `[inherited]`

7.51.4.3 `void` `spot::ltl::clone_visitor::visit` ([constant](#) \* `c`)

7.51.4.4 `void` `spot::ltl::clone_visitor::visit` ([multop](#) \* `mo`)

7.51.4.5 `void` `spot::ltl::clone_visitor::visit` ([binop](#) \* `bo`)

7.51.4.6 `void` `spot::ltl::clone_visitor::visit` ([unop](#) \* `uo`)

7.51.4.7 `void` `spot::ltl::clone_visitor::visit` ([atomic\\_prop](#) \* `ap`)

7.51.4.8 `void` `spot::ltl::simplify_f_g_visitor::visit` ([binop](#) \* `bo`) `[virtual]`

Reimplemented from [spot::ltl::clone\\_visitor](#).

### 7.51.5 Member Data Documentation

7.51.5.1 [formula](#)\* `spot::ltl::clone_visitor::result_` `[protected, inherited]`

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

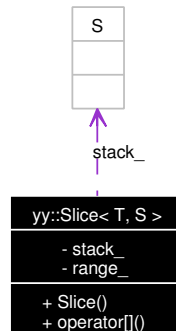
- [ltlvisit/simpfg.hh](#)



## 7.52 yy::Slice< T, S > Class Template Reference

```
#include <stack.hh>
```

Collaboration diagram for yy::Slice< T, S >:



### Public Member Functions

- [Slice](#) (const S &stack, unsigned range)
- const T & [operator\[\]](#) (unsigned i) const

### Private Attributes

- const S & [stack\\_](#)
- unsigned [range\\_](#)

```
template<class T, class S = Stack< T >> class yy::Slice< T, S >
```

### 7.52.1 Constructor & Destructor Documentation

**7.52.1.1** `template<class T, class S = Stack< T >> yy::Slice< T, S >::Slice (const S & stack, unsigned range)` `[inline]`

### 7.52.2 Member Function Documentation

**7.52.2.1** `template<class T, class S = Stack< T >> const T& yy::Slice< T, S >::operator[] (unsigned i) const` `[inline]`

### 7.52.3 Member Data Documentation

**7.52.3.1** `template<class T, class S = Stack< T >> unsigned yy::Slice< T, S >::range\_` `[private]`

**7.52.3.2** `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:

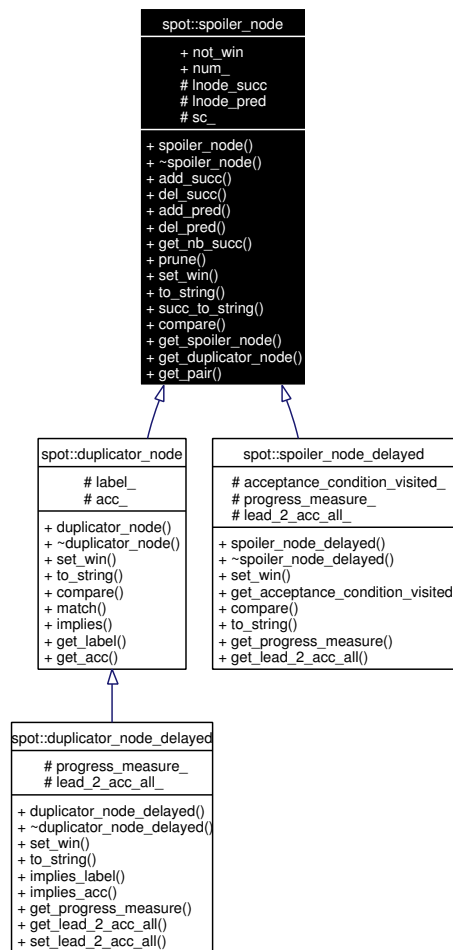
- [ltparse/stack.hh](#)

## 7.53 spot::spoiler\_node Class Reference

Spoiler node of parity game graph.

```
#include <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\\_](#)

#### 7.53.1 Detailed Description

Spoiler node of parity game graph.

#### 7.53.2 Constructor & Destructor Documentation

**7.53.2.1** `spot::spoiler_node::spoiler_node (const state * d_node, const state * s_node, int num)`

**7.53.2.2** `virtual spot::spoiler_node::~spoiler\_node ()` [virtual]

#### 7.53.3 Member Function Documentation

**7.53.3.1** `virtual void spot::spoiler_node::add_pred (spoiler\_node * n)` [virtual]

**7.53.3.2** `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.

**7.53.3.3** `virtual bool spot::spoiler_node::compare (spoiler\_node * n)` [virtual]

Reimplemented in [spot::duplicator\\_node](#), and [spot::spoiler\\_node\\_delayed](#).

**7.53.3.4** `virtual void spot::spoiler_node::del_pred ()` [virtual]

**7.53.3.5** `void spot::spoiler_node::del_succ (spoiler\_node * n)`

7.53.3.6 `const state* spot::spoiler_node::get_duplicator_node ()`

7.53.3.7 `int spot::spoiler_node::get_nb_succ ()`

7.53.3.8 `state\_couple* spot::spoiler_node::get_pair ()`

7.53.3.9 `const state* spot::spoiler_node::get_spoiler_node ()`

7.53.3.10 `bool spot::spoiler_node::prune ()`

7.53.3.11 `virtual bool spot::spoiler_node::set_win ()` [virtual]

Reimplemented in [spot::duplicator\\_node](#), [spot::spoiler\\_node\\_delayed](#), and [spot::duplicator\\_node\\_delayed](#).

7.53.3.12 `virtual std::string spot::spoiler_node::succ_to_string ()` [virtual]

7.53.3.13 `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](#).

## 7.53.4 Member Data Documentation

7.53.4.1 `sn\_v* spot::spoiler_node::lnode_pred` [protected]

7.53.4.2 `sn\_v* spot::spoiler_node::lnode_succ` [protected]

7.53.4.3 `bool spot::spoiler_node::not_win`

7.53.4.4 `int spot::spoiler_node::num_`

7.53.4.5 `state\_couple* spot::spoiler_node::sc_` [protected]

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

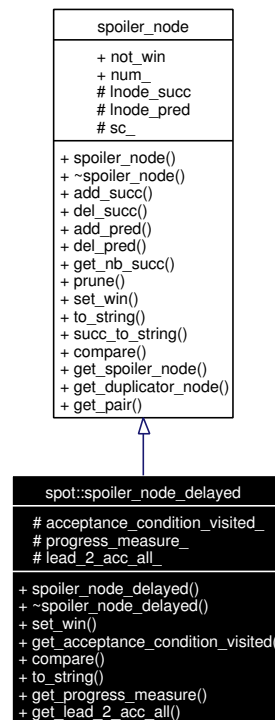
- [tgbaalgos/reductgba\\_sim.hh](#)

## 7.54 spot::spoiler\_node\_delayed Class Reference

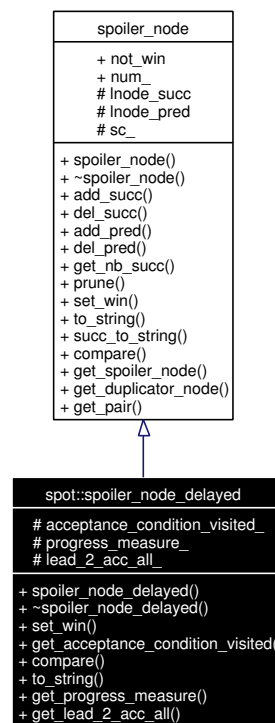
Spoiler node of parity game graph for delayed simulation.

```
#include <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, bool l2a=true)
- [~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 [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 [acceptance\\_condition\\_visited\\_](#)
- int [progress\\_measure\\_](#)
- bool [lead\\_2\\_acc\\_all\\_](#)
- [sn\\_v](#) \* [lnode\\_succ](#)
- [sn\\_v](#) \* [lnode\\_pred](#)
- [state\\_couple](#) \* [sc\\_](#)

#### 7.54.1 Detailed Description

Spoiler node of parity game graph for delayed simulation.

#### 7.54.2 Constructor & Destructor Documentation

**7.54.2.1** spot::spoiler\_node\_delayed::spoiler\_node\_delayed (const [state](#) \* d\_node, const [state](#) \* s\_node, bdd a, int num, bool l2a = true)

**7.54.2.2** spot::spoiler\_node\_delayed::~~spoiler\_node\_delayed ()

### 7.54.3 Member Function Documentation

**7.54.3.1** `virtual void spot::spoiler_node::add_pred (spoiler_node * n) [virtual, inherited]`

**7.54.3.2** `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.

**7.54.3.3** `virtual bool spot::spoiler_node_delayed::compare (spoiler_node * n) [virtual]`

Reimplemented from `spot::spoiler_node`.

**7.54.3.4** `virtual void spot::spoiler_node::del_pred () [virtual, inherited]`

**7.54.3.5** `void spot::spoiler_node::del_succ (spoiler_node * n) [inherited]`

**7.54.3.6** `bdd spot::spoiler_node_delayed::get_acceptance_condition_visited () const`

**7.54.3.7** `const state* spot::spoiler_node::get_duplicator_node () [inherited]`

**7.54.3.8** `bool spot::spoiler_node_delayed::get_lead_2_acc_all ()`

**7.54.3.9** `int spot::spoiler_node::get_nb_succ () [inherited]`

**7.54.3.10** `state_couple* spot::spoiler_node::get_pair () [inherited]`

**7.54.3.11** `int spot::spoiler_node_delayed::get_progress_measure () const`

**7.54.3.12** `const state* spot::spoiler_node::get_spoiler_node () [inherited]`

**7.54.3.13** `bool spot::spoiler_node::prune () [inherited]`

**7.54.3.14** `bool spot::spoiler_node_delayed::set_win () [virtual]`

Return true if the progress\_measure has changed.

Reimplemented from `spot::spoiler_node`.

**7.54.3.15** `virtual std::string spot::spoiler_node::succ_to_string () [virtual, inherited]`

**7.54.3.16** `virtual std::string spot::spoiler_node_delayed::to_string (const tgba * a) [virtual]`

Reimplemented from `spot::spoiler_node`.

### 7.54.4 Member Data Documentation

**7.54.4.1** `bdd spot::spoiler_node_delayed::acceptance_condition_visited_` [protected]

a Bdd for retain all the acceptance condition that a node has visited.

**7.54.4.2** `bool spot::spoiler_node_delayed::lead_2_acc_all_` [protected]

**7.54.4.3** `sn_v* spot::spoiler_node::lnode_pred` [protected, inherited]

**7.54.4.4** `sn_v* spot::spoiler_node::lnode_succ` [protected, inherited]

**7.54.4.5** `bool spot::spoiler_node::not_win` [inherited]

**7.54.4.6** `int spot::spoiler_node::num_` [inherited]

**7.54.4.7** `int spot::spoiler_node_delayed::progress_measure_` [protected]

**7.54.4.8** `state_couple* spot::spoiler_node::sc_` [protected, inherited]

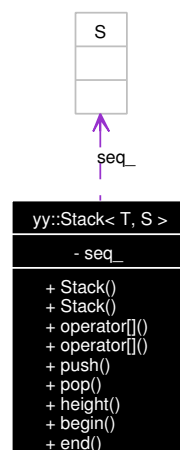
The documentation for this class was generated from the following file:

- [tgbaalgos/reductgba\\_sim.hh](#)

## 7.55 yy::Stack< T, S > Class Template Reference

```
#include <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 n)
- T & [operator\[\]](#) (unsigned i)
- const T & [operator\[\]](#) (unsigned i) const
- void [push](#) (const T &t)
- void [pop](#) (unsigned n=1)
- unsigned [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 >
```

**7.55.1 Member Typedef Documentation**

**7.55.1.1** `template<class T, class S = std::deque< T >> typedef S::const_iterator yy::Stack< T, S >::`[ConstIterator](#)

**7.55.1.2** `template<class T, class S = std::deque< T >> typedef S::iterator yy::Stack< T, S >::`[Iterator](#)

**7.55.2 Constructor & Destructor Documentation**

**7.55.2.1** `template<class T, class S = std::deque< T >> yy::Stack< T, S >::`[Stack](#) () `[inline]`

**7.55.2.2** `template<class T, class S = std::deque< T >> yy::Stack< T, S >::`[Stack](#) (unsigned n) `[inline]`

**7.55.3 Member Function Documentation**

**7.55.3.1** `template<class T, class S = std::deque< T >> ConstIterator yy::Stack< T, S >::`[begin](#) () `const [inline]`

**7.55.3.2** `template<class T, class S = std::deque< T >> ConstIterator yy::Stack< T, S >::`[end](#) () `const [inline]`

**7.55.3.3** `template<class T, class S = std::deque< T >> unsigned yy::Stack< T, S >::`[height](#) () `const [inline]`

**7.55.3.4** ] `template<class T, class S = std::deque< T >> const T& yy::Stack< T, S >::operator[] (unsigned i) const` [inline]

**7.55.3.5** ] `template<class T, class S = std::deque< T >> T& yy::Stack< T, S >::operator[] (unsigned i)` [inline]

**7.55.3.6** `template<class T, class S = std::deque< T >> void yy::Stack< T, S >::pop (unsigned n = 1)` [inline]

**7.55.3.7** `template<class T, class S = std::deque< T >> void yy::Stack< T, S >::push (const T & t)` [inline]

## 7.55.4 Member Data Documentation

**7.55.4.1** `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:

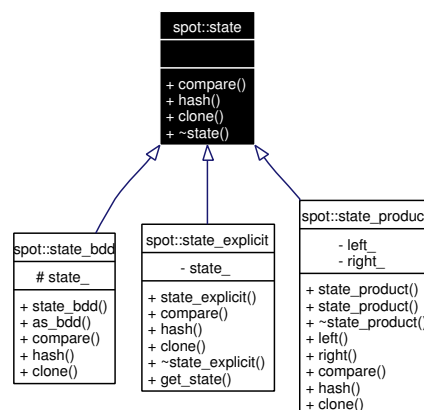
- [Itlparse/stack.hh](#)

## 7.56 spot::state Class Reference

Abstract class for states.

`#include <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](#) ()

### 7.56.1 Detailed Description

Abstract class for states.

### 7.56.2 Constructor & Destructor Documentation

**7.56.2.1** virtual [spot::state::~~state](#) () [inline, virtual]

### 7.56.3 Member Function Documentation

**7.56.3.1** virtual [state](#)\* [spot::state::clone](#) () const [pure virtual]

Duplicate a state.

Implemented in [spot::state\\_bdd](#), [spot::state\\_explicit](#), and [spot::state\\_product](#).

**7.56.3.2** 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](#).

**7.56.3.3** 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\\_bdd](#), [spot::state\\_explicit](#), and [spot::state\\_product](#).

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

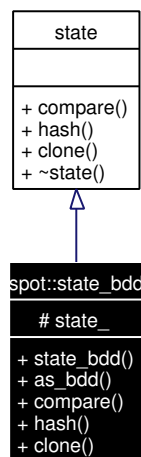
- [tgba/state.hh](#)

## 7.57 spot::state\_bdd Class Reference

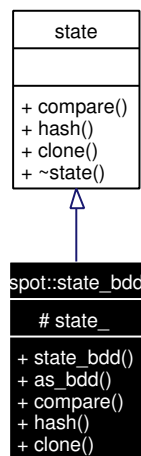
A state whose representation is a BDD.

```
#include <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.*

#### 7.57.1 Detailed Description

A state whose representation is a BDD.

#### 7.57.2 Constructor & Destructor Documentation

**7.57.2.1** `spot::state_bdd::state_bdd (bdd s) [inline]`

#### 7.57.3 Member Function Documentation

**7.57.3.1** `virtual bdd spot::state_bdd::as_bdd () const [inline, virtual]`

Return the BDD part of the state.

**7.57.3.2** `virtual state\_bdd* spot::state_bdd::clone () const [virtual]`

Duplicate a state.

Implements [spot::state](#).

**7.57.3.3** `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](#).

**7.57.3.4 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 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](#).

**7.57.4 Member Data Documentation****7.57.4.1 bdd spot::state\_bdd::state\_ [protected]**

BDD representation of the state.

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

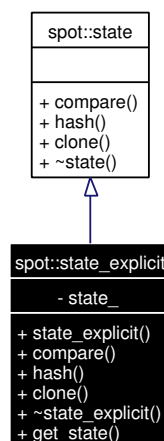
- [tgba/statebdd.hh](#)

**7.58 spot::state\_explicit Class Reference**

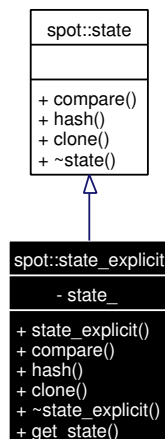
States used by [spot::tgba\\_explicit](#).

```
#include <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_`

#### 7.58.1 Detailed Description

States used by `spot::tgba_explicit`.

#### 7.58.2 Constructor & Destructor Documentation

**7.58.2.1** `spot::state_explicit::state_explicit` (const `tgba_explicit::state` \*s) [inline]

**7.58.2.2** `virtual spot::state_explicit::~~state_explicit` () [inline, virtual]

### 7.58.3 Member Function Documentation

#### 7.58.3.1 virtual [state\\_explicit](#)\* spot::state\_explicit::clone () const [virtual]

Duplicate a state.

Implements [spot::state](#).

#### 7.58.3.2 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](#).

#### 7.58.3.3 virtual int spot::state\_explicit::compare (const [spot::state](#) \* *other*) const [virtual]

#### 7.58.3.4 const [tgba\\_explicit::state](#)\* spot::state\_explicit::get\_state () const

#### 7.58.3.5 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](#).

### 7.58.4 Member Data Documentation

#### 7.58.4.1 const [tgba\\_explicit::state](#)\* spot::state\_explicit::state\_ [private]

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

- [tgba/tgbaexplicit.hh](#)

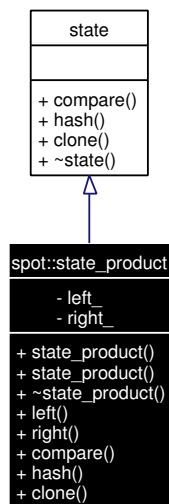


## 7.59 spot::state\_product Class Reference

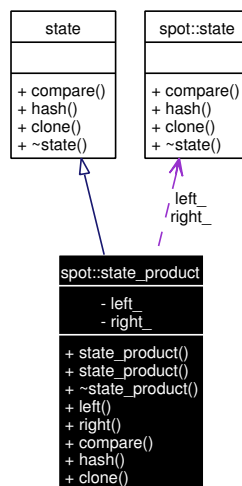
A state for [spot::tgbaproduct](#).

```
#include <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.*

### 7.59.1 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.

### 7.59.2 Constructor & Destructor Documentation

#### 7.59.2.1 `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.

#### 7.59.2.2 `spot::state_product::state_product (const state_product & o)`

Copy constructor.

#### 7.59.2.3 `virtual spot::state_product::~~state_product()` [virtual]

### 7.59.3 Member Function Documentation

#### 7.59.3.1 `virtual state_product* spot::state_product::clone () const` [virtual]

Duplicate a state.

Implements [spot::state](#).

#### 7.59.3.2 `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](#).

#### 7.59.3.3 `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](#).

#### 7.59.3.4 `state* spot::state_product::left () const` [inline]

#### 7.59.3.5 `state* spot::state_product::right () const` [inline]

### 7.59.4 Member Data Documentation

#### 7.59.4.1 `state* spot::state_product::left_` [private]

State from the left automaton.

#### 7.59.4.2 `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](#)

## 7.60 spot::state\_ptr\_equal Struct Reference

An Equivalence Relation for `state*`.

```
#include <state.hh>
```

### Public Member Functions

- `bool operator()` (const `state` \*left, const `state` \*right) const

#### 7.60.1 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_less_than,  
             spot::state_ptr_equal> seen;
```

#### 7.60.2 Member Function Documentation

**7.60.2.1** `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](#)

## 7.61 spot::state\_ptr\_hash Struct Reference

Hash Function for `state*`.

```
#include <state.hh>
```

### Public Member Functions

- `size_t operator()` (const `state` \*that) const

#### 7.61.1 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_less_than,  
             spot::state_ptr_equal> seen;
```

### 7.61.2 Member Function Documentation

#### 7.61.2.1 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](#)

## 7.62 spot::state\_ptr\_less\_than Struct Reference

Strict Weak Ordering for state\*.

```
#include <state.hh>
```

### Public Member Functions

- bool operator() (const state \*left, const state \*right) const

#### 7.62.1 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;
```

### 7.62.2 Member Function Documentation

#### 7.62.2.1 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](#)

## 7.63 spot::string\_hash Struct Reference

A hash function for strings.

```
#include <hash.hh>
```

### Public Member Functions

- size\_t operator() (const std::string &s) const

#### 7.63.1 Detailed Description

A hash function for strings.

### 7.63.2 Member Function Documentation

#### 7.63.2.1 size\_t spot::string\_hash::operator() (const std::string & s) const [inline]

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

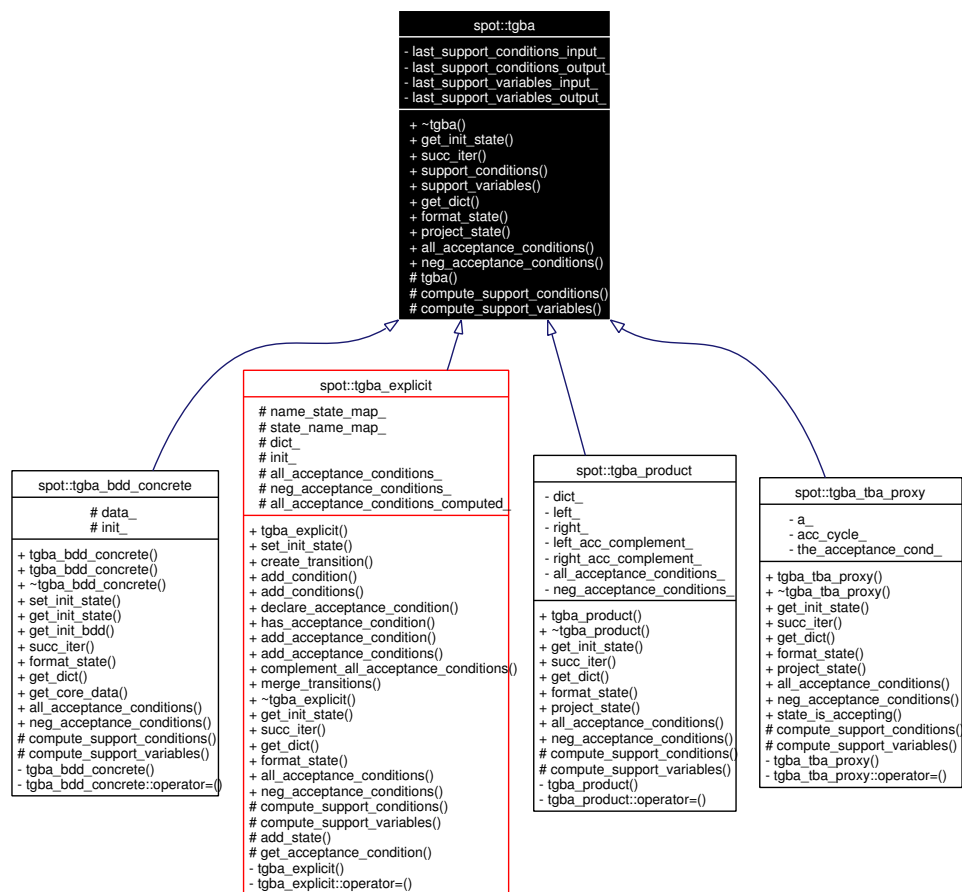
- [misc/hash.hh](#)

## 7.64 spot::tgba Class Reference

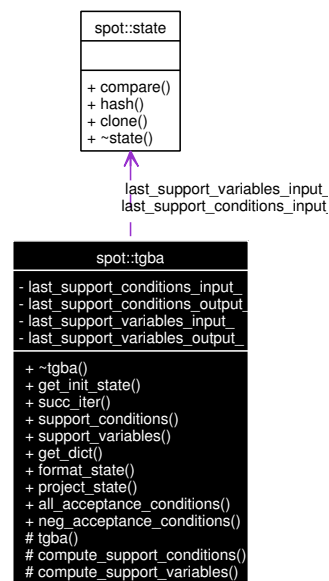
A Transition-based Generalized Büchi Automaton.

```
#include <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 `state * project_state` (const `state *s`, const `tgba *t`) const  
*Project a state on an automata.*
- virtual bdd `all_acceptance_conditions` () const =0  
*Return the set of all acceptance conditions used by this automaton.*
- 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\\_](#)

#### 7.64.1 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.

#### 7.64.2 Constructor & Destructor Documentation

**7.64.2.1** `spot::tgba::tgba()` [protected]

**7.64.2.2** `virtual spot::tgba::~~tgba()` [virtual]

#### 7.64.3 Member Function Documentation

**7.64.3.1** `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](#).



**7.64.3.2 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](#).

**7.64.3.3 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](#).

**7.64.3.4 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](#).

**7.64.3.5 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](#).

**7.64.3.6 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](#).

**7.64.3.7 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](#).

**7.64.3.8 virtual state\* spot::tgba::project\_state (const state \* s, const tgba \* t) const** [virtual]

Project a state on an automata.

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`.

**7.64.3.9** `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`.

**7.64.3.10** `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.

**7.64.3.11** `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.

#### 7.64.4 Member Data Documentation

7.64.4.1 `const state* spot::tgba::last_support_conditions_input_` [mutable, private]

7.64.4.2 `bdd spot::tgba::last_support_conditions_output_` [mutable, private]

7.64.4.3 `const state* spot::tgba::last_support_variables_input_` [mutable, private]

7.64.4.4 `bdd spot::tgba::last_support_variables_output_` [mutable, private]

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

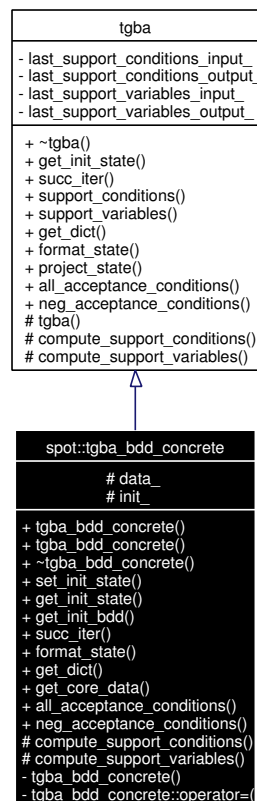
- [tgba/tgba.hh](#)

## 7.65 spot::tgba\_bdd\_concrete Class Reference

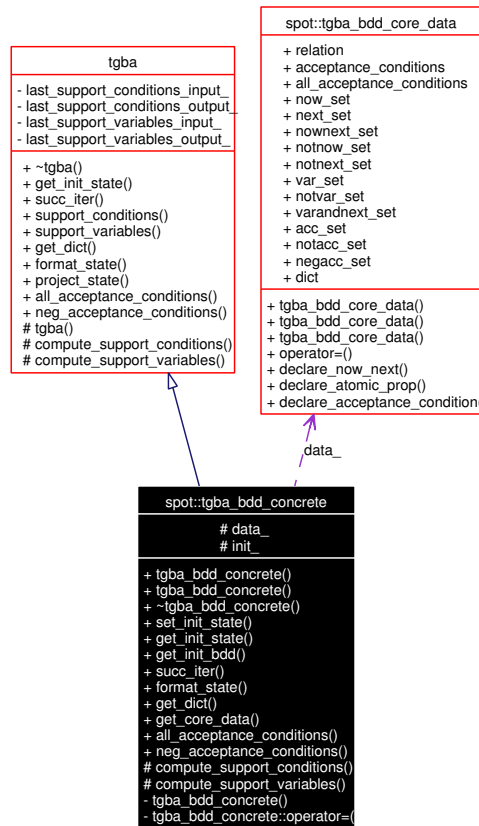
A concrete `spot::tgba` implemented using BDDs.

```
#include <tgba_bdd_concrete.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 [state](#) \* [project\\_state](#) (const [state](#) \*s, const [tgba](#) \*t) const  
*Project a state on an automata.*

### 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](#) &)

### 7.65.1 Detailed Description

A concrete [spot::tgba](#) implemented using BDDs.

### 7.65.2 Constructor & Destructor Documentation

#### 7.65.2.1 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.

#### 7.65.2.2 spot::tgba\_bdd\_concrete::tgba\_bdd\_concrete (const [tgba\\_bdd\\_factory](#) & *fact*, bdd *init*)

Construct a [tgba\\_bdd\\_concrete](#) with known initial state.

#### 7.65.2.3 virtual spot::tgba\_bdd\_concrete::~~tgba\_bdd\_concrete () [virtual]

#### 7.65.2.4 spot::tgba\_bdd\_concrete::tgba\_bdd\_concrete (const [tgba\\_bdd\\_concrete](#) &) [private]

### 7.65.3 Member Function Documentation

#### 7.65.3.1 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](#).

#### 7.65.3.2 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](#).

#### 7.65.3.3 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](#).

#### 7.65.3.4 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](#).

**7.65.3.5 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.

**7.65.3.6 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](#).

**7.65.3.7 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;
```

**7.65.3.8 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](#).

**7.65.3.9 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](#).

**7.65.3.10 virtual state\* spot::tgba::project\_state (const state \*s, const tgba \*t) const [virtual, inherited]**

Project a state on an automata.

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](#).

**7.65.3.11 virtual void spot::tgba\_bdd\_concrete::set\_init\_state (bdd s) [virtual]**

Set the initial state.

**7.65.3.12 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](#).

**7.65.3.13 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.

**7.65.3.14 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.



**7.65.3.15** [tgba\\_bdd\\_concrete&](#) [spot::tgba\\_bdd\\_concrete::tgba\\_bdd\\_concrete::operator=](#) (const [tgba\\_bdd\\_concrete](#) &) [[private](#)]

#### 7.65.4 Member Data Documentation

**7.65.4.1** [tgba\\_bdd\\_core\\_data](#) [spot::tgba\\_bdd\\_concrete::data\\_](#) [[protected](#)]

Core data associated to the automaton.

**7.65.4.2** [bdd](#) [spot::tgba\\_bdd\\_concrete::init\\_](#) [[protected](#)]

Initial state.

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

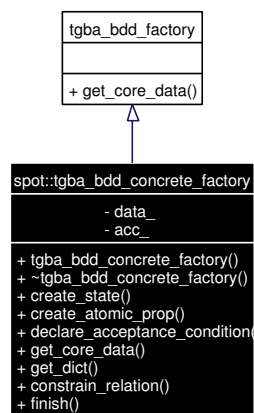
- [tgba/tgbabddconcrete.hh](#)

## 7.66 spot::tgba\_bdd\_concrete\_factory Class Reference

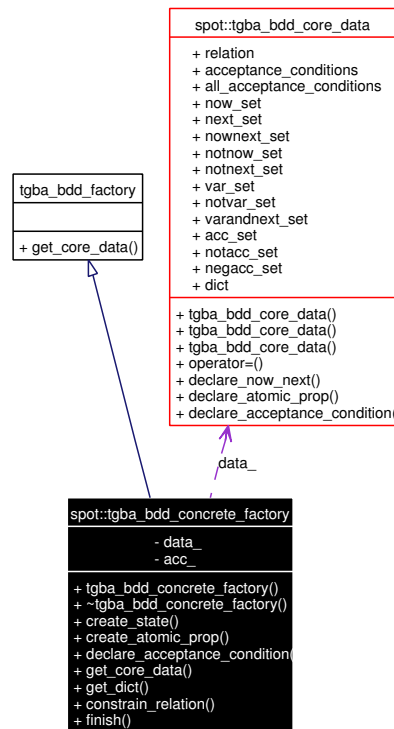
Helper class to build a [spot::tgba\\_bdd\\_concrete](#) object.

```
#include <tgbabddconcretefactory.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.*

### 7.66.1 Detailed Description

Helper class to build a [spot::tgba\\_bdd\\_concrete](#) object.

### 7.66.2 Member Typedef Documentation

**7.66.2.1** `typedef Sgi::hash_map<const ltl::formula*, bdd, ptr_hash<ltl::formula> > spot::tgba\_bdd\_concrete\_factory::acc\_map\_ [private]`

### 7.66.3 Constructor & Destructor Documentation

**7.66.3.1** `spot::tgba\_bdd\_concrete\_factory::tgba\_bdd\_concrete\_factory (bdd\_dict * dict)`

**7.66.3.2** `virtual spot::tgba\_bdd\_concrete\_factory::~tgba\_bdd\_concrete\_factory () [virtual]`

### 7.66.4 Member Function Documentation

**7.66.4.1** `void spot::tgba\_bdd\_concrete\_factory::constrain\_relation (bdd new_rel)`

Add a new constraint to the relation.

**7.66.4.2** `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()`.

**7.66.4.3** `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()`.

**7.66.4.4** `void spot::tgba_bdd_concrete_factory::declare_acceptance_condition (bdd b, const ltd::formula * a)`

Declare an acceptance condition.

Formula such as 'f 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

**7.66.4.5** `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()`.

**7.66.4.6** `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](#).

**7.66.4.7** `bdd\_dict* spot::tgba_bdd_concrete_factory::get_dict () const`

## 7.66.5 Member Data Documentation

**7.66.5.1** `acc\_map spot::tgba\_bdd\_concrete\_factory::acc\_ [private]`

BDD associated to each acceptance condition.

**7.66.5.2** `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:

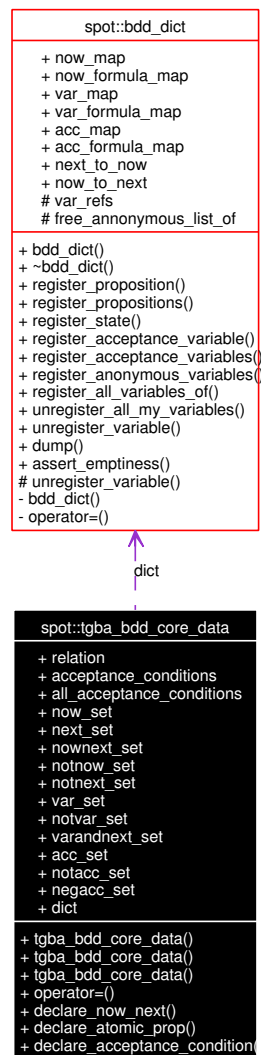
- [tgba/tgbabddconcretefactory.hh](#)

## 7.67 spot::tgba\_bdd\_core\_data Struct Reference

Core data for a TGBA encoded using BDDs.

```
#include <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 automic 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.*

### 7.67.1 Detailed Description

Core data for a TGBA encoded using BDDs.

### 7.67.2 Constructor & Destructor Documentation

#### 7.67.2.1 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.

#### 7.67.2.2 spot::tgba\_bdd\_core\_data::tgba\_bdd\_core\_data (const [tgba\\_bdd\\_core\\_data](#) & *copy*)

Copy constructor.

#### 7.67.2.3 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.

### 7.67.3 Member Function Documentation

#### 7.67.3.1 void spot::tgba\_bdd\_core\_data::declare\_acceptance\_condition (*bdd prom*)

Update the variable sets to take a new acceptance condition into account.

#### 7.67.3.2 void spot::tgba\_bdd\_core\_data::declare\_atomic\_prop (*bdd var*)

Update the variable sets to take a new atomic proposition into account.

#### 7.67.3.3 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.

#### 7.67.3.4 const [tgba\\_bdd\\_core\\_data](#)& spot::tgba\_bdd\_core\_data::operator= (const [tgba\\_bdd\\_core\\_data](#) & *copy*)

### 7.67.4 Member Data Documentation

#### 7.67.4.1 [bdd](#) spot::tgba\_bdd\_core\_data::acc\_set

The (positive) conjunction of all variables which are acceptance conditions.

**7.67.4.2 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  $Acc[a] \& \neg Acc[b] \& \neg Acc[c]$ . If the transition is in both  $a$  and  $b$ , we'll return  $(Acc[a] \& \neg Acc[b] \& \neg Acc[c]) \mid (\neg Acc[a] \& Acc[b] \& \neg Acc[c])$ .

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.

**7.67.4.3 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.

**7.67.4.4 bdd dict\* spot::tgba\_bdd\_core\_data::dict**

The dictionary used by the automata.

**7.67.4.5 bdd spot::tgba\_bdd\_core\_data::negacc\_set**

The negative conjunction of all variables which are acceptance conditions.

**7.67.4.6 bdd spot::tgba\_bdd\_core\_data::next\_set**

The conjunction of all Next variables, in their positive form.

**7.67.4.7 bdd spot::tgba\_bdd\_core\_data::notacc\_set**

The (positive) conjunction of all variables which are not acceptance conditions.

**7.67.4.8 bdd spot::tgba\_bdd\_core\_data::notnext\_set**

The (positive) conjunction of all variables which are not Next variables.



**7.67.4.9** `bdd spot::tgba_bdd_core_data::notnow_set`

The (positive) conjunction of all variables which are not Now variables.

**7.67.4.10** `bdd spot::tgba_bdd_core_data::notvar_set`

The (positive) conjunction of all variables which are not atomic propositions.

**7.67.4.11** `bdd spot::tgba_bdd_core_data::now_set`

The conjunction of all Now variables, in their positive form.

**7.67.4.12** `bdd spot::tgba_bdd_core_data::nownext_set`

The conjunction of all Now and Next variables, in their positive form.

**7.67.4.13** `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

**7.67.4.14** `bdd spot::tgba_bdd_core_data::var_set`

The (positive) conjunction of all variables which are atomic propositions.

**7.67.4.15** `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:

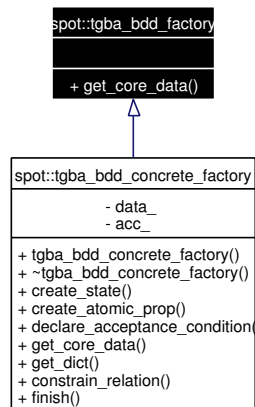
- [tgba/tgbabddcoredata.hh](#)

**7.68** `spot::tgba_bdd_factory` Class Reference

Abstract class for `spot::tgba_bdd_concrete` factories.

```
#include <tgbabddfactory.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.*

### 7.68.1 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.

### 7.68.2 Member Function Documentation

**7.68.2.1** 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:

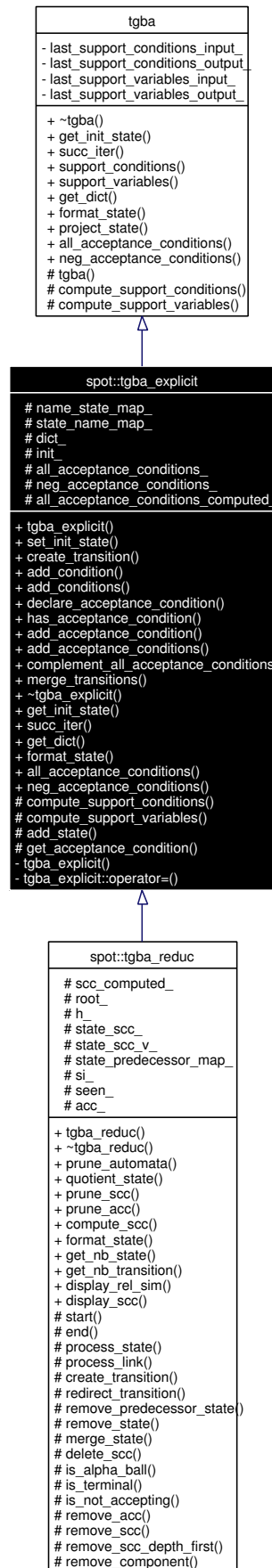
- [tgba/tgababddfactory.hh](#)

## 7.69 spot::tgba\_explicit Class Reference

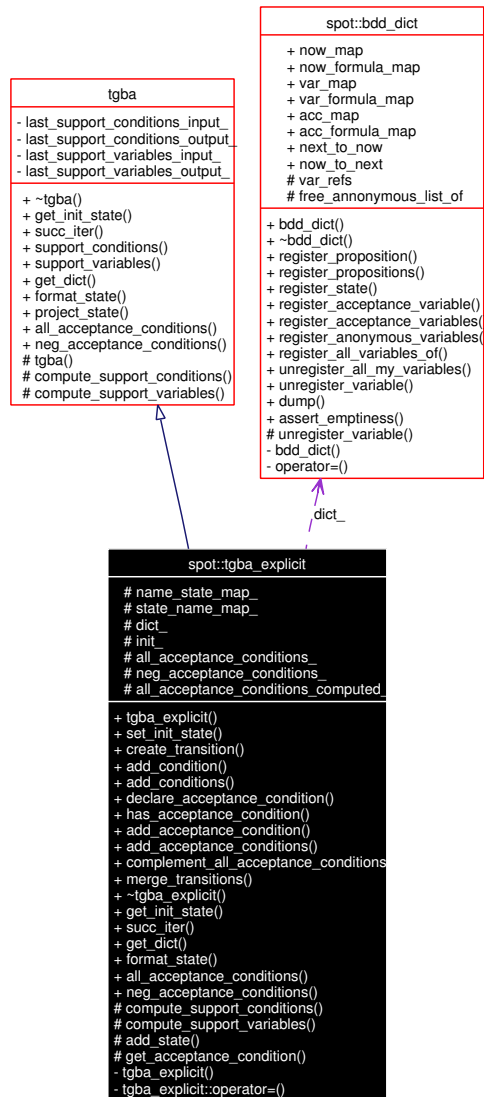
Explicit representation of a [spot::tgba](#).

```
#include <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)
- void [set\\_init\\_state](#) (const std::string &state)
- [transition](#) \* [create\\_transition](#) (const std::string &source, const std::string &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)
- 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` ()
- 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 `state` \* `project_state` (const `state` \*s, const `tgba` \*t) const  
*Project a state on an automata.*

## 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
- [state](#) \* [add\\_state](#) (const std::string &name)
- 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)

#### 7.69.1 Detailed Description

Explicit representation of a [spot::tgba](#).

#### 7.69.2 Member Typedef Documentation

**7.69.2.1** typedef [Sgi::hash\\_map](#)<const std::string, [tgba\\_explicit::state](#)\*, [string\\_hash](#)> [spot::tgba\\_explicit::ns\\_map](#) [protected]

**7.69.2.2** typedef [Sgi::hash\\_map](#)<const [tgba\\_explicit::state](#)\*, std::string, [ptr\\_hash](#)<[tgba\\_explicit::state](#)>> [spot::tgba\\_explicit::sn\\_map](#) [protected]

**7.69.2.3** typedef std::list<[transition](#)\*> [spot::tgba\\_explicit::state](#)

#### 7.69.3 Constructor & Destructor Documentation

**7.69.3.1** [spot::tgba\\_explicit::tgba\\_explicit](#) ([bdd\\_dict](#) \* dict)

**7.69.3.2** virtual [spot::tgba\\_explicit::~tgba\\_explicit](#) () [virtual]

**7.69.3.3** spot::tgba\_explicit::tgba\_explicit (const [tgba\\_explicit](#) & *other*) [`private`]

## 7.69.4 Member Function Documentation

**7.69.4.1** void spot::tgba\_explicit::add\_acceptance\_condition ([transition](#) \* *t*, const [ltl::formula](#) \* *f*)

**7.69.4.2** void spot::tgba\_explicit::add\_acceptance\_conditions ([transition](#) \* *t*, [bdd](#) *f*)

This assumes that all acceptance conditions in *f* are known from dict.

**7.69.4.3** void spot::tgba\_explicit::add\_condition ([transition](#) \* *t*, const [ltl::formula](#) \* *f*)

**7.69.4.4** void spot::tgba\_explicit::add\_conditions ([transition](#) \* *t*, [bdd](#) *f*)

This assumes that all variables in *f* are known from dict.

**7.69.4.5** [state](#)\* spot::tgba\_explicit::add\_state (const [std::string](#) & *name*) [`protected`]

**7.69.4.6** 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](#).

**7.69.4.7** void spot::tgba\_explicit::complement\_all\_acceptance\_conditions ()

**7.69.4.8** 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](#).

**7.69.4.9** virtual [bdd](#) spot::tgba\_explicit::compute\_support\_conditions (const [spot::state](#) \* *state*) const [`protected`, `virtual`]

**7.69.4.10** 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](#).

**7.69.4.11** virtual [bdd](#) spot::tgba\_explicit::compute\_support\_variables (const [spot::state](#) \* *state*) const [`protected`, `virtual`]

**7.69.4.12** [transition\\*](#) `spot::tgba_explicit::create_transition (const std::string & source, const std::string & dest)`

**7.69.4.13** `void spot::tgba_explicit::declare_acceptance_condition (const ltl::formula * f)`

**7.69.4.14** `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](#).

**7.69.4.15** `virtual std::string spot::tgba_explicit::format_state (const spot::state * state) const` `[virtual]`

Reimplemented in [spot::tgba\\_reduc](#).

**7.69.4.16** `bdd spot::tgba_explicit::get_acceptance_condition (const ltl::formula * f)` `[protected]`

**7.69.4.17** `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](#).

**7.69.4.18** `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](#).

**7.69.4.19** `bool spot::tgba_explicit::has_acceptance_condition (const ltl::formula * f) const`

**7.69.4.20** `void spot::tgba_explicit::merge_transitions ()`

**7.69.4.21** `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](#).

**7.69.4.22** `virtual state* spot::tgba::project_state (const state * s, const tgba * t) const` [virtual, inherited]

Project a state on an automata.

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](#).

**7.69.4.23** `void spot::tgba_explicit::set_init_state (const std::string & state)`

**7.69.4.24** `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 synchronized 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](#).

**7.69.4.25** `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]

**7.69.4.26** `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.

#### 7.69.4.27 `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.

#### 7.69.4.28 `tgba\_explicit& spot::tgba_explicit::tgba_explicit::operator= (const tgba\_explicit & other)` [private]

### 7.69.5 Member Data Documentation

#### 7.69.5.1 `bdd spot::tgba\_explicit::all\_acceptance\_conditions\_` [mutable, protected]

#### 7.69.5.2 `bool spot::tgba\_explicit::all\_acceptance\_conditions\_computed\_` [mutable, protected]

#### 7.69.5.3 `bdd\_dict\* spot::tgba\_explicit::dict\_` [protected]

#### 7.69.5.4 `tgba\_explicit::state\* spot::tgba\_explicit::init\_` [protected]

#### 7.69.5.5 `ns\_map spot::tgba\_explicit::name\_state\_map\_` [protected]

#### 7.69.5.6 `bdd spot::tgba\_explicit::neg\_acceptance\_conditions\_` [protected]

#### 7.69.5.7 `sn\_map spot::tgba\_explicit::state\_name\_map\_` [protected]

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

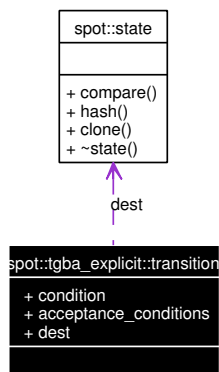
- [tgba/tgbaexplicit.hh](#)

## 7.70 spot::tgba\_explicit::transition Struct Reference

Explicit transitions (used by [spot::tgba\\_explicit](#)).

```
#include <tgbaexplicit.hh>
```

Collaboration diagram for `spot::tgba_explicit::transition`:



### Public Attributes

- bdd [condition](#)
- bdd [acceptance\\_conditions](#)
- [state](#) \* [dest](#)

### 7.70.1 Detailed Description

Explicit transitions (used by [spot::tgba\\_explicit](#)).

### 7.70.2 Member Data Documentation

#### 7.70.2.1 bdd [spot::tgba\\_explicit::transition::acceptance\\_conditions](#)

#### 7.70.2.2 bdd [spot::tgba\\_explicit::transition::condition](#)

#### 7.70.2.3 [state\\*](#) [spot::tgba\\_explicit::transition::dest](#)

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

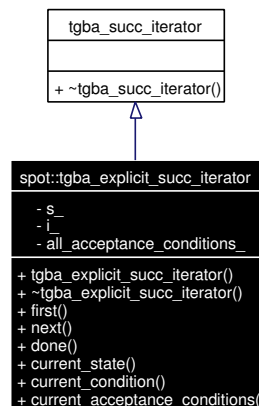
- [tgba/tgbaexplicit.hh](#)

## 7.71 spot::tgba\_explicit\_succ\_iterator Class Reference

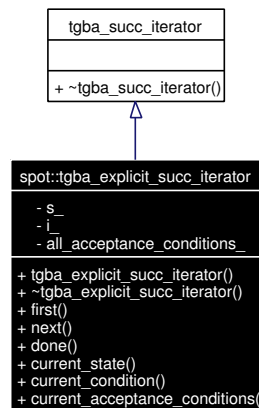
Successor iterators used by [spot::tgba\\_explicit](#).

```
#include <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\\_](#)

### 7.71.1 Detailed Description

Successor iterators used by [spot::tgba\\_explicit](#).

### 7.71.2 Constructor & Destructor Documentation

**7.71.2.1** [spot::tgba\\_explicit\\_succ\\_iterator::tgba\\_explicit\\_succ\\_iterator](#) (const [tgba\\_explicit::state](#) \* [s](#), bdd [all\\_acc](#))

**7.71.2.2** virtual [spot::tgba\\_explicit\\_succ\\_iterator::~~tgba\\_explicit\\_succ\\_iterator](#) () [inline, virtual]

### 7.71.3 Member Function Documentation

**7.71.3.1** 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](#).

**7.71.3.2** 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](#).

**7.71.3.3** 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](#).

**7.71.3.4 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](#).

**7.71.3.5 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](#).

**7.71.3.6 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](#).

**7.71.4 Member Data Documentation****7.71.4.1 bdd spot::tgba\_explicit\_succ\_iterator::all\_acceptance\_conditions\_** [private]**7.71.4.2 tgba\_explicit::state::const\_iterator spot::tgba\_explicit\_succ\_iterator::i\_** [private]**7.71.4.3 const tgba\_explicit::state\* spot::tgba\_explicit\_succ\_iterator::s\_** [private]

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

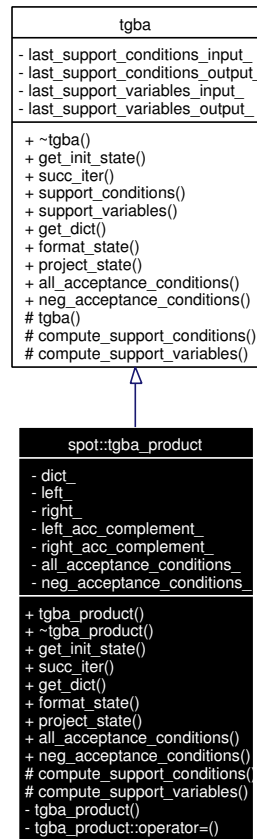
- [tgba/tgbaexplicit.hh](#)

**7.72 spot::tgba\_product Class Reference**

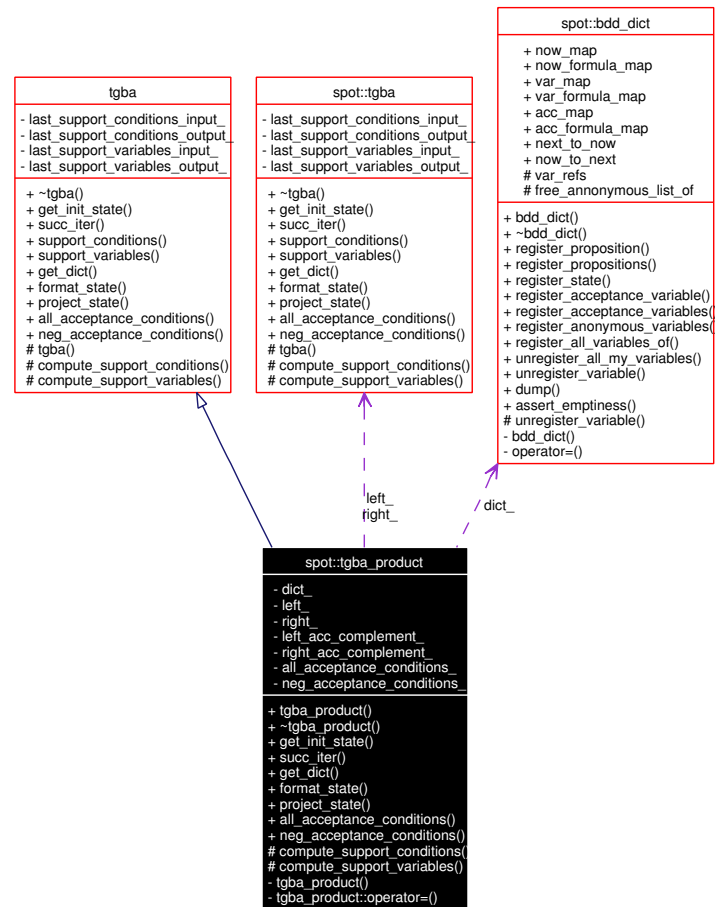
A lazy product. (States are computed on the fly.).

```
#include <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 [state](#) \* [project\\_state](#) (const [state](#) \*s, const [tgba](#) \*t) const  
*Project a state on an automata.*



- 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.*

### 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\\_](#)

#### 7.72.1 Detailed Description

A lazy product. (States are computed on the fly.).

#### 7.72.2 Constructor & Destructor Documentation

##### 7.72.2.1 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.

**7.72.2.2** `virtual spot::tgba_product::~~tgba_product () [virtual]`

**7.72.2.3** `spot::tgba_product::tgba_product (const tgba_product &) [private]`

### 7.72.3 Member Function Documentation

**7.72.3.1** `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](#).

**7.72.3.2** `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](#).

**7.72.3.3** `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](#).

**7.72.3.4** `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](#).

**7.72.3.5** `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](#).

**7.72.3.6 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](#).

**7.72.3.7 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](#).

**7.72.3.8 virtual state\* spot::tgba\_product::project\_state (const state \* s, const tgba \* t) const [virtual]**

Project a state on an automata.

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 from [spot::tgba](#).

**7.72.3.9 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](#).

**7.72.3.10 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.

**7.72.3.11 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.

**7.72.3.12 [tgba\\_product](#)& spot::tgba\_product::tgba\_product::operator= (const [tgba\\_product](#) &)** [private]**7.72.4 Member Data Documentation****7.72.4.1 bdd spot::tgba\_product::all\_acceptance\_conditions\_** [private]**7.72.4.2 [bdd\\_dict](#)\* spot::tgba\_product::dict\_** [private]**7.72.4.3 const [tgba](#)\* spot::tgba\_product::left\_** [private]**7.72.4.4 bdd spot::tgba\_product::left\_acc\_complement\_** [private]**7.72.4.5 bdd spot::tgba\_product::neg\_acceptance\_conditions\_** [private]**7.72.4.6 const [tgba](#)\* spot::tgba\_product::right\_** [private]**7.72.4.7 bdd spot::tgba\_product::right\_acc\_complement\_** [private]

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

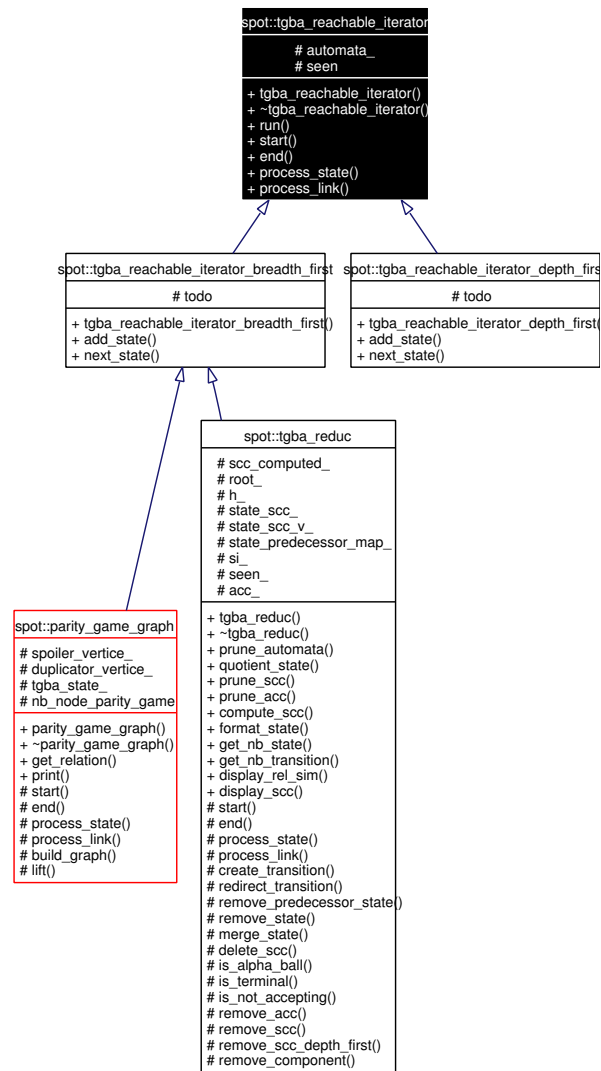
- [tgba/tgbaproduct.hh](#)

## 7.73 spot::tgba\_reachable\_iterator Class Reference

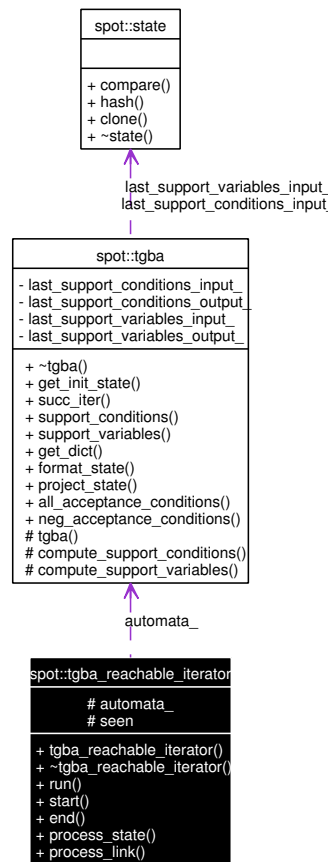
Iterate over all reachable states of a [spot::tgba](#).

```
#include <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](#) (int in, 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.*

#### 7.73.1 Detailed Description

Iterate over all reachable states of a [spot::tgba](#).

#### 7.73.2 Member Typedef Documentation

**7.73.2.1** 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](#).

#### 7.73.3 Constructor & Destructor Documentation

**7.73.3.1** [spot::tgba\\_reachable\\_iterator::tgba\\_reachable\\_iterator](#) (const [tgba](#) \* *a*)

**7.73.3.2** virtual [spot::tgba\\_reachable\\_iterator::~~tgba\\_reachable\\_iterator](#) () [virtual]

#### 7.73.4 Member Function Documentation

**7.73.4.1** 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](#).

**7.73.4.2** 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](#).

**7.73.4.3 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](#).

**7.73.4.4 virtual void spot::tgba\_reachable\_iterator::process\_link (int *in*, int *out*, const [tgba\\_succ\\_iterator](#) \* *si*)** [virtual]

Called by [run\(\)](#) to process a transition.

**Parameters:**

- in* The source state number.
- out* The destination state number.
- si* The [spot::tgba\\_succ\\_iterator](#) positionned on the current transition.

Reimplemented in [spot::tgba\\_reduc](#), and [spot::parity\\_game\\_graph](#).

**7.73.4.5 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* An unique number assigned to *s*.
- si* The [spot::tgba\\_succ\\_iterator](#) for *s*.

Reimplemented in [spot::parity\\_game\\_graph](#).

**7.73.4.6 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.

**7.73.4.7 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](#).

**7.73.5 Member Data Documentation****7.73.5.1 const [tgba](#)\* spot::tgba\_reachable\_iterator::automata\_** [protected]

The [spot::tgba](#) to explore.



### 7.73.5.2 seen\_map spot::tgba\_reachable\_iterator::seen [protected]

States already seen.

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

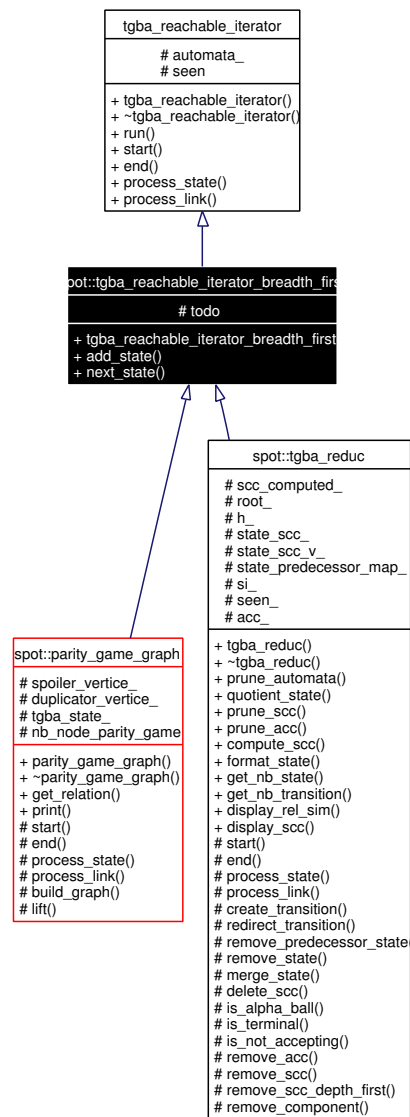
- tgbaalgos/reachiter.hh

## 7.74 spot::tgba\_reachable\_iterator\_breadth\_first Class Reference

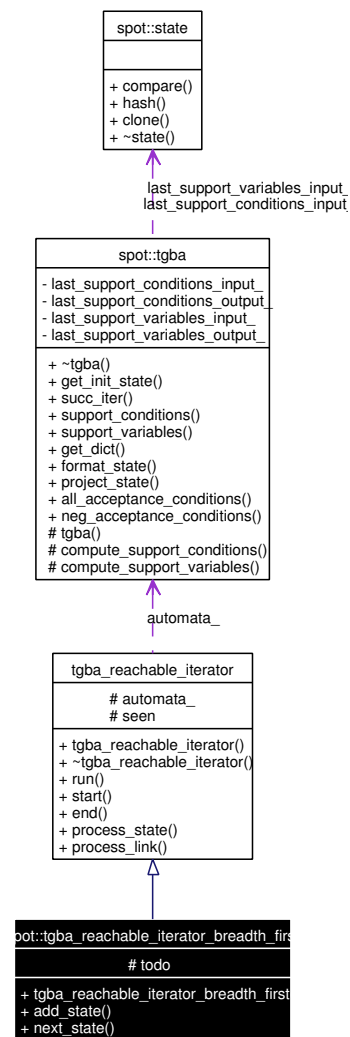
An implementation of [spot::tgba\\_reachable\\_iterator](#) that browses states breadth first.

```
#include <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](#) (int in, 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.*

## 7.74.1 Detailed Description

An implementation of [spot::tgba\\_reachable\\_iterator](#) that browses states breadth first.

## 7.74.2 Member Typedef Documentation

**7.74.2.1** 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](#).

## 7.74.3 Constructor & Destructor Documentation

**7.74.3.1** [spot::tgba\\_reachable\\_iterator\\_breadth\\_first::tgba\\_reachable\\_iterator\\_breadth\\_first](#) (const [tgba](#) \* *a*)

## 7.74.4 Member Function Documentation

**7.74.4.1** virtual void [spot::tgba\\_reachable\\_iterator\\_breadth\\_first::add\\_state](#) (const [state](#) \* *s*) [virtual]

Implements [spot::tgba\\_reachable\\_iterator](#).

**7.74.4.2** 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](#).

**7.74.4.3** `virtual const state* spot::tgba_reachable_iterator_breadth_first::next_state ()` [virtual]

Called by `run()` to obtain the.

Implements `spot::tgba_reachable_iterator`.

**7.74.4.4** `virtual void spot::tgba_reachable_iterator::process_link (int in, int out, const tgba_succ_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::tgba_succ_iterator` positionned on the current transition.

Reimplemented in `spot::tgba_reduc`, and `spot::parity_game_graph`.

**7.74.4.5** `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* An unique number assigned to *s*.

*si* The `spot::tgba_succ_iterator` for *s*.

Reimplemented in `spot::parity_game_graph`.

**7.74.4.6** `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.

**7.74.4.7** `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`.

## 7.74.5 Member Data Documentation

**7.74.5.1** `const tgba* spot::tgba_reachable_iterator::automata_` [protected, inherited]

The `spot::tgba` to explore.

**7.74.5.2** `seen_map spot::tgba_reachable_iterator::seen` [protected, inherited]

States already seen.

### 7.74.5.3 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:

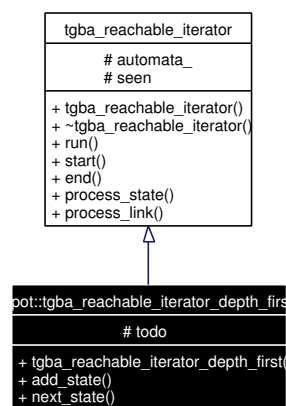
- tgbaalgos/reachiter.hh

## 7.75 spot::tgba\_reachable\_iterator\_depth\_first Class Reference

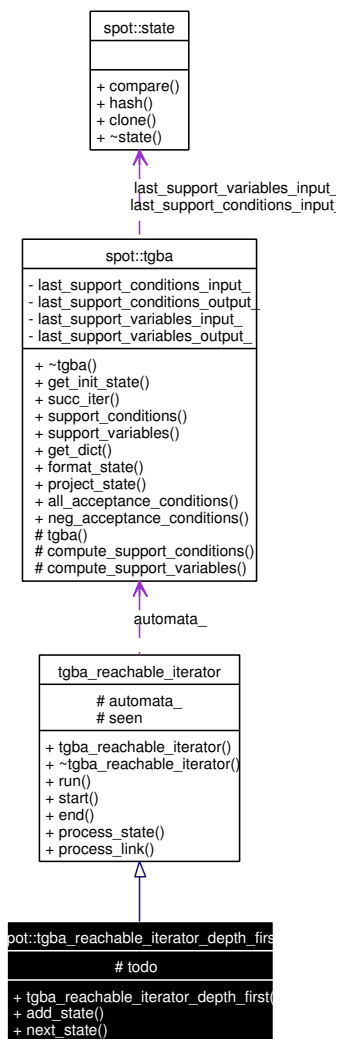
An implementation of [spot::tgba\\_reachable\\_iterator](#) that browses states depth first.

```
#include <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](#) (int in, 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.*

## 7.75.1 Detailed Description

An implementation of [spot::tgba\\_reachable\\_iterator](#) that browses states depth first.

## 7.75.2 Member Typedef Documentation

**7.75.2.1** 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](#).

## 7.75.3 Constructor & Destructor Documentation

**7.75.3.1** [spot::tgba\\_reachable\\_iterator\\_depth\\_first::tgba\\_reachable\\_iterator\\_depth\\_first](#) (const [tgba](#) \* a)

## 7.75.4 Member Function Documentation

**7.75.4.1** virtual void [spot::tgba\\_reachable\\_iterator\\_depth\\_first::add\\_state](#) (const [state](#) \* s) [virtual]

Implements [spot::tgba\\_reachable\\_iterator](#).

**7.75.4.2** 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](#).

**7.75.4.3** `virtual const state* spot::tgba_reachable_iterator_depth_first::next_state ()` [virtual]

Called by `run()` to obtain the.

Implements `spot::tgba_reachable_iterator`.

**7.75.4.4** `virtual void spot::tgba_reachable_iterator::process_link (int in, int out, const tgba_succ_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::tgba_succ_iterator` positionned on the current transition.

Reimplemented in `spot::tgba_reduc`, and `spot::parity_game_graph`.

**7.75.4.5** `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* An unique number assigned to *s*.

*si* The `spot::tgba_succ_iterator` for *s*.

Reimplemented in `spot::parity_game_graph`.

**7.75.4.6** `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.

**7.75.4.7** `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`.

## 7.75.5 Member Data Documentation

**7.75.5.1** `const tgba* spot::tgba_reachable_iterator::automata_` [protected, inherited]

The `spot::tgba` to explore.

**7.75.5.2** `seen_map spot::tgba_reachable_iterator::seen` [protected, inherited]

States already seen.



**7.75.5.3** `std::stack<const state*>` `spot::tgba_reachable_iterator_depth_first::todo`  
[protected]

A stack of states yet to explore.

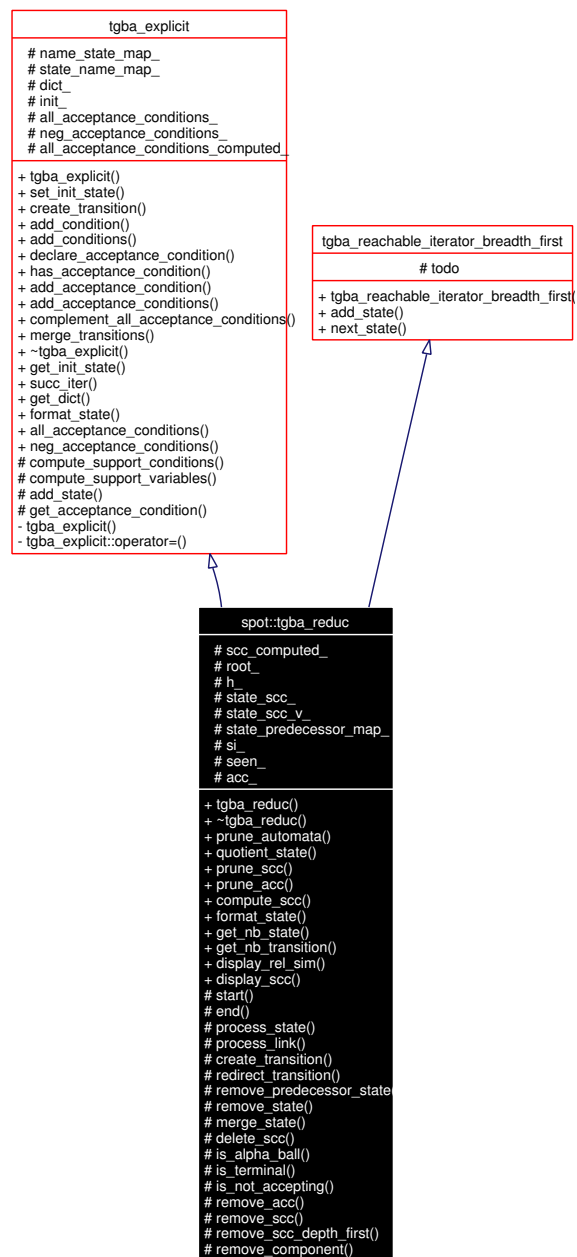
The documentation for this class was generated from the following file:

- `tgbaalgos/reachiter.hh`

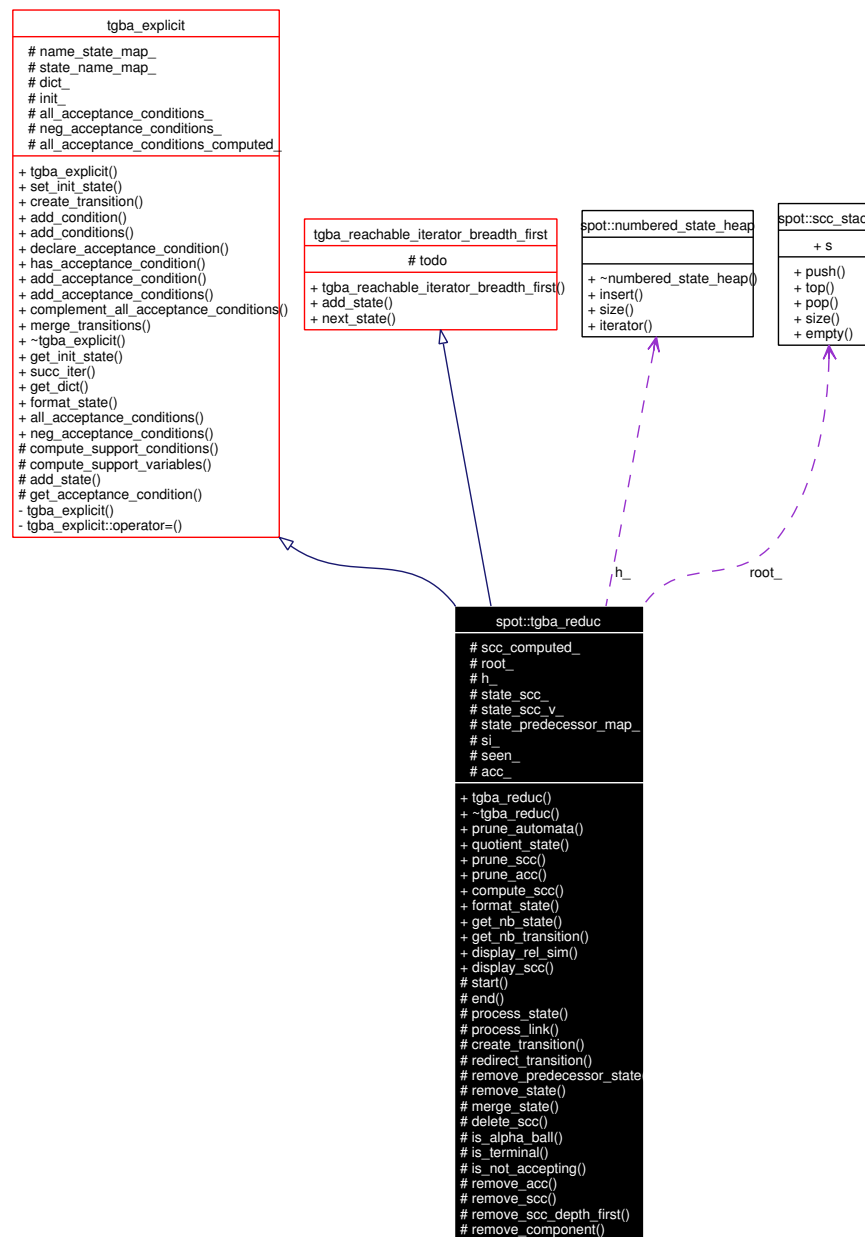
## 7.76 spot::tgba\_reduc Class Reference

```
#include <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 [prune\\_automata](#) ([simulation\\_relation](#) \*rel)
- void [quotient\\_state](#) ([simulation\\_relation](#) \*rel)

- 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.*
- int `get_nb_state` ()  
*Obsolete.*
- int `get_nb_transition` ()
- void `display_rel_sim` (simulation\_relation \*rel, std::ostream &os)
- void `display_scc` (std::ostream &os)
- void `set_init_state` (const std::string &state)
- transition \* `create_transition` (const std::string &source, const std::string &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)
- 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` ()
- 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 `state` \* `project_state` (const `state` \*s, const `tgba` \*t) const  
*Project a state on an automata.*
- 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)

### 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 `delete_scc` ()

- bool [is\\_alpha\\_ball](#) (const [spot::state](#) \*s, bdd label=bddfalse, int n=-1)
- 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\\_scc\\_depth\\_first](#) ([spot::state](#) \*s, int n=-1)  
*Same as remove\_scc but more efficient.*
- void [remove\\_component](#) (const [spot::state](#) \*from)  
*For compute\_scc.*
- 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().*
- [state](#) \* [add\\_state](#) (const std::string &name)
- 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.*

### 7.76.1 Member Typedef Documentation

**7.76.1.1** typedef Sgi::hash\_map<const std::string, tgba\_explicit::state\*, string\_hash> spot::tgba\_explicit::ns\_map [protected, inherited]

**7.76.1.2** 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.

**7.76.1.3** typedef Sgi::hash\_map<const tgba\_explicit::state\*, std::string, ptr\_hash<tgba\_explicit::state>> spot::tgba\_explicit::sn\_map [protected, inherited]

**7.76.1.4** typedef Sgi::hash\_map<const tgba\_explicit::state\*, std::list<state\*>\*, ptr\_hash<tgba\_explicit::state>> spot::tgba\_reduc::sp\_map [protected]

**7.76.1.5** typedef std::list<transition\*> spot::tgba\_explicit::state [inherited]

### 7.76.2 Constructor & Destructor Documentation

**7.76.2.1** spot::tgba\_reduc::tgba\_reduc (const tgba \* *a*, const numbered\_state\_heap\_factory \* *nshf* = numbered\_state\_heap\_hash\_map\_factory::instance())

**7.76.2.2** spot::tgba\_reduc::~~tgba\_reduc ()

### 7.76.3 Member Function Documentation

**7.76.3.1** void spot::tgba\_explicit::add\_acceptance\_condition (transition \* *t*, const ltl::formula \* *f*) [inherited]

**7.76.3.2** void spot::tgba\_explicit::add\_acceptance\_conditions (transition \* *t*, bdd *f*) [inherited]

This assumes that all acceptance conditions in *f* are known from dict.

**7.76.3.3** void spot::tgba\_explicit::add\_condition (transition \* *t*, const ltl::formula \* *f*) [inherited]

**7.76.3.4** void spot::tgba\_explicit::add\_conditions (transition \* *t*, bdd *f*) [inherited]

This assumes that all variables in *f* are known from dict.

**7.76.3.5** virtual void spot::tgba\_reachable\_iterator\_breadth\_first::add\_state (const state \* *s*) [virtual, inherited]

Implements spot::tgba\_reachable\_iterator.

**7.76.3.6** `state*` `spot::tgba_explicit::add_state` (`const std::string & name`) [`protected`, `inherited`]

**7.76.3.7** `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](#).

**7.76.3.8** `void` `spot::tgba_explicit::complement_all_acceptance_conditions` () [`inherited`]

**7.76.3.9** `void` `spot::tgba_reduc::compute_scc` ()

Compute the maximal SCC of the automata.

**7.76.3.10** `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](#).

**7.76.3.11** `virtual bdd` `spot::tgba_explicit::compute_support_conditions` (`const spot::state * state`) `const` [`protected`, `virtual`, `inherited`]

**7.76.3.12** `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](#).

**7.76.3.13** `virtual bdd` `spot::tgba_explicit::compute_support_variables` (`const spot::state * state`) `const` [`protected`, `virtual`, `inherited`]

**7.76.3.14** `transition*` `spot::tgba_explicit::create_transition` (`const std::string & source`, `const std::string & dest`) [`inherited`]

**7.76.3.15** `transition*` `spot::tgba_reduc::create_transition` (`const spot::state * source`, `const spot::state * dest`) [`protected`]

Create a transition using two state of a TGBA.

**7.76.3.16** `void` `spot::tgba_explicit::declare_acceptance_condition` (`const ltl::formula * f`) [`inherited`]



**7.76.3.17 void spot::tgba\_reduc::delete\_scc ()** [protected]

Remove all the scc which are terminal and doesn't contains all the acceptance conditions.

**7.76.3.18 void spot::tgba\_reduc::display\_rel\_sim (simulation\_relation \* rel, std::ostream & os)****7.76.3.19 void spot::tgba\_reduc::display\_scc (std::ostream & os)****7.76.3.20 void spot::tgba\_reduc::end ()** [protected, virtual]

Called by [run\(\)](#) once all states have been explored.

Reimplemented from [spot::tgba\\_reachable\\_iterator](#).

**7.76.3.21 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](#).

**7.76.3.22 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](#).

**7.76.3.23 bdd spot::tgba\_explicit::get\_acceptance\_condition (const ltl::formula \* f)** [protected, inherited]**7.76.3.24 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](#).

**7.76.3.25 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](#).

**7.76.3.26 int spot::tgba\_reduc::get\_nb\_state ()**

Obsolete.

**7.76.3.27** `int spot::tgba_reduc::get_nb_transition ()`

**7.76.3.28** `bool spot::tgba_explicit::has_acceptance_condition (const ltl::formula * f) const` [inherited]

**7.76.3.29** `bool spot::tgba_reduc::is_alpha_ball (const spot::state * s, bdd label = bddfalse, 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}
}
```

**7.76.3.30** `bool spot::tgba_reduc::is_not_accepting (const spot::state * s, int n = -1) [protected]`

**7.76.3.31** `bool spot::tgba_reduc::is_terminal (const spot::state * s, int n = -1) [protected]`

**7.76.3.32** `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.

**7.76.3.33** `void spot::tgba_explicit::merge_transitions () [inherited]`

**7.76.3.34** `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](#).

**7.76.3.35** `virtual const state\* spot::tgba_reachable_iterator_breadth_first::next_state () [virtual, inherited]`

Called by [run\(\)](#) to obtain the.

Implements [spot::tgba\\_reachable\\_iterator](#).

**7.76.3.36** `void spot::tgba_reduc::process_link (int in, int out, const tgba\_succ\_iterator * si)` [protected, virtual]

Called by [run\(\)](#) to process a transition.

**Parameters:**

*in* The source state number.

*out* The destination state number.

*si* The [spot::tgba\\_succ\\_iterator](#) positionned on the current transition.

Reimplemented from [spot::tgba\\_reachable\\_iterator](#).

**7.76.3.37** `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* An unique number assigned to *s*.

*si* The [spot::tgba\\_succ\\_iterator](#) for *s*.

Reimplemented in [spot::parity\\_game\\_graph](#).

**7.76.3.38** `void spot::tgba_reduc::process_state (const spot::state * s, int n, tgba\_succ\_iterator * si)` [protected]

**7.76.3.39** `virtual state* spot::tgba::project_state (const state * s, const tgba * t) const` [virtual, inherited]

Project a state on an automata.

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](#).

**7.76.3.40** `void spot::tgba_reduc::prune_acc ()`

Remove some useless accepting condition.

**7.76.3.41** `void spot::tgba_reduc::prune_automata (simulation\_relation * rel)`

Reduce the automata using a relation simulation Do not call this method with a delayed simulation relation.

**7.76.3.42 void spot::tgba\_reduc::prune\_scc ()**

Remove all state which not lead to an accepting cycle.

**7.76.3.43 void spot::tgba\_reduc::quotient\_state (simulation\_relation \* rel)**

Build the quotient automata. Call this method when use to a delayed simulation relation.

**7.76.3.44 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.

**7.76.3.45 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.

**7.76.3.46 void spot::tgba\_reduc::remove\_component (const spot::state \* from) [protected]**

For compute\_scc.

**7.76.3.47 void spot::tgba\_reduc::remove\_predecessor\_state (const state \* s, const state \* p) [protected]**

Remove p of the predecessor of s.

**7.76.3.48 void spot::tgba\_reduc::remove\_scc (spot::state \* s) [protected]**

Remove all the state which belong to the same scc that s.

**7.76.3.49 void spot::tgba\_reduc::remove\_scc\_depth\_first (spot::state \* s, int n = -1) [protected]**

Same as remove\_scc but more efficient.

**7.76.3.50 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.

**7.76.3.51 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.

**7.76.3.52 void spot::tgba\_explicit::set\_init\_state (const std::string & state) [inherited]****7.76.3.53 void spot::tgba\_reduc::start () [protected, virtual]**

Called by [run\(\)](#) before starting its iteration.

Reimplemented from [spot::tgba\\_reachable\\_iterator](#).

**7.76.3.54** 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](#).

**7.76.3.55** 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]

**7.76.3.56** 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.

**7.76.3.57** 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.

## 7.76.4 Member Data Documentation

**7.76.4.1** bdd [spot::tgba\\_reduc::acc\\_](#) [protected]

**7.76.4.2** `bdd` `spot::tgba_explicit::all_acceptance_conditions_` [mutable, protected, inherited]

**7.76.4.3** `bool` `spot::tgba_explicit::all_acceptance_conditions_computed_` [mutable, protected, inherited]

**7.76.4.4** `const tgba*` `spot::tgba_reachable_iterator::automata_` [protected, inherited]

The `spot::tgba` to explore.

**7.76.4.5** `bdd_dict*` `spot::tgba_explicit::dict_` [protected, inherited]

**7.76.4.6** `numbered_state_heap*` `spot::tgba_reduc::h_` [protected]

**7.76.4.7** `tgba_explicit::state*` `spot::tgba_explicit::init_` [protected, inherited]

**7.76.4.8** `ns_map` `spot::tgba_explicit::name_state_map_` [protected, inherited]

**7.76.4.9** `bdd` `spot::tgba_explicit::neg_acceptance_conditions_` [protected, inherited]

**7.76.4.10** `scc_stack` `spot::tgba_reduc::root_` [protected]

**7.76.4.11** `bool` `spot::tgba_reduc::scc_computed_` [protected]

**7.76.4.12** `seen_map` `spot::tgba_reachable_iterator::seen_` [protected, inherited]

States already seen.

**7.76.4.13** `seen_map*` `spot::tgba_reduc::seen_` [protected]

**7.76.4.14** `seen_map` `spot::tgba_reduc::si_` [protected]

**7.76.4.15** `sn_map` `spot::tgba_explicit::state_name_map_` [protected, inherited]

**7.76.4.16** `sp_map` `spot::tgba_reduc::state_predecessor_map_` [protected]

**7.76.4.17** `std::stack<const spot::state*>` `spot::tgba_reduc::state_scc_` [protected]

**7.76.4.18** `Sgi::hash_map<int, const spot::state*>` `spot::tgba_reduc::state_scc_v_` [protected]

#### 7.76.4.19 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

## 7.77 spot::tgba\_statistics Struct Reference

```
#include <stats.hh>
```

### Public Attributes

- unsigned [transitions](#)
- unsigned [states](#)

### 7.77.1 Member Data Documentation

#### 7.77.1.1 unsigned spot::tgba\_statistics::states

#### 7.77.1.2 unsigned spot::tgba\_statistics::transitions

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

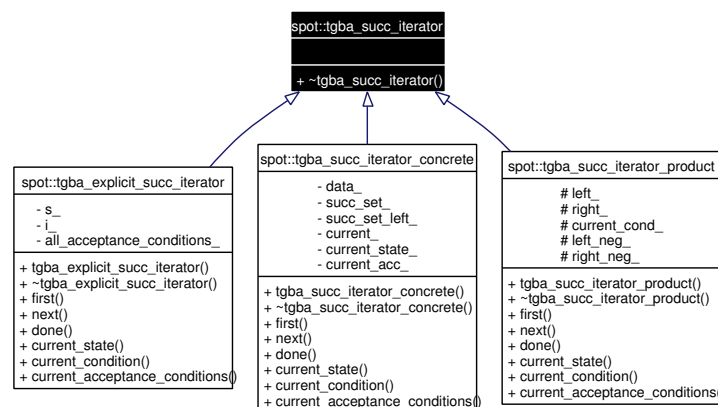
- tgbaalgos/stats.hh

## 7.78 spot::tgba\_succ\_iterator Class Reference

Iterate over the successors of a state.

```
#include <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.*

**7.78.1 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.

**7.78.2 Constructor & Destructor Documentation**

**7.78.2.1** virtual `spot::tgba_succ_iterator::~~tgba_succ_iterator()` [inline, virtual]

**7.78.3 Member Function Documentation**

**7.78.3.1** 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`.



**7.78.3.2** `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`.

**7.78.3.3** `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`.

**7.78.3.4** `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`.

**7.78.3.5** `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`.

**7.78.3.6** `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:

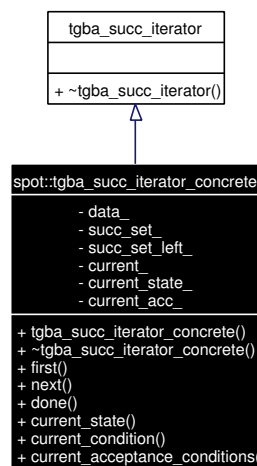
- [tgba/succiter.hh](#)

## 7.79 spot::tgba\_succ\_iterator\_concrete Class Reference

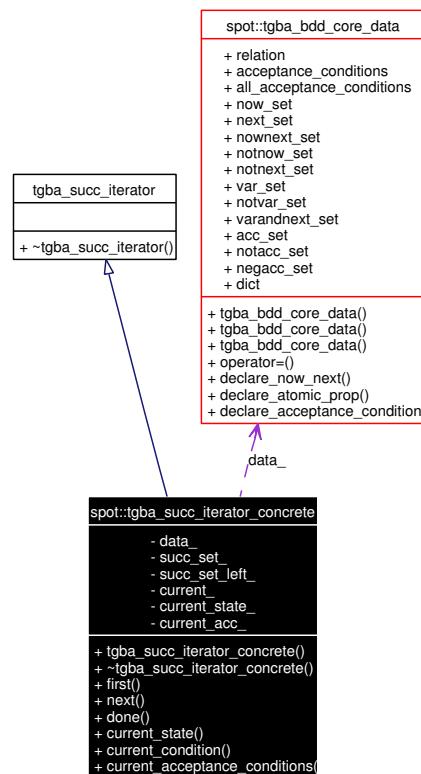
A concrete iterator over successors of a TGBA state.

```
#include <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.*

### 7.79.1 Detailed Description

A concrete iterator over successors of a TGBA state.

### 7.79.2 Constructor & Destructor Documentation

#### 7.79.2.1 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.

**7.79.2.2** virtual [spot::tgba\\_succ\\_iterator\\_concrete::~~tgba\\_succ\\_iterator\\_concrete](#) ()  
[virtual]

### 7.79.3 Member Function Documentation

**7.79.3.1** 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](#).

**7.79.3.2** `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](#).

**7.79.3.3** `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](#).

**7.79.3.4** `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](#).

**7.79.3.5** `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](#).

**7.79.3.6** `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](#).

**7.79.4 Member Data Documentation****7.79.4.1** `bdd spot::tgba_succ_iterator_concrete::current_` [private]

Current successor, as a conjunction of atomic proposition and Next variables.

**7.79.4.2** bdd `spot::tgba_succ_iterator_concrete::current_acc_` [private]

Accepting conditions for the current transition.

**7.79.4.3** bdd `spot::tgba_succ_iterator_concrete::current_state_` [private]

Current successor, as a conjunction of Now variables.

**7.79.4.4** const `tgba_bdd_core_data& spot::tgba_succ_iterator_concrete::data_` [private]

Core data of the automaton.

**7.79.4.5** bdd `spot::tgba_succ_iterator_concrete::succ_set_` [private]

The set of successors.

**7.79.4.6** 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:

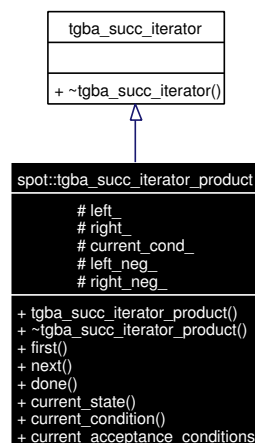
- [tgba/succiterconcrete.hh](#)

**7.80 spot::tgba\_succ\_iterator\_product Class Reference**

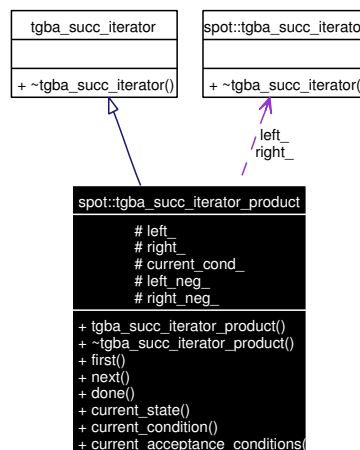
Iterate over the successors of a product computed on the fly.

```
#include <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_()`

### 7.80.1 Detailed Description

Iterate over the successors of a product computed on the fly.

### 7.80.2 Constructor & Destructor Documentation

**7.80.2.1** `spot::tgba_succ_iterator_product::tgba_succ_iterator_product (tgba_succ_iterator * left, tgba_succ_iterator * right, bdd left_neg, bdd right_neg)`

**7.80.2.2** `virtual spot::tgba_succ_iterator_product::~~tgba_succ_iterator_product ()` [virtual]

### 7.80.3 Member Function Documentation

**7.80.3.1** `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`.

**7.80.3.2** `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`.

**7.80.3.3** `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`.

**7.80.3.4** `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](#).

#### 7.80.3.5 `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](#).

#### 7.80.3.6 `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](#).

#### 7.80.3.7 `void spot::tgba_succ_iterator_product::next_non_false_()` [private]

#### 7.80.3.8 `void spot::tgba_succ_iterator_product::step_()` [private]

Internal routines to advance to the next successor.

### 7.80.4 Member Data Documentation

#### 7.80.4.1 `bdd spot::tgba_succ_iterator_product::current_cond_` [protected]

#### 7.80.4.2 `tgba_succ_iterator* spot::tgba_succ_iterator_product::left_` [protected]

#### 7.80.4.3 `bdd spot::tgba_succ_iterator_product::left_neg_` [protected]

#### 7.80.4.4 `tgba_succ_iterator* spot::tgba_succ_iterator_product::right_` [protected]

#### 7.80.4.5 `bdd spot::tgba_succ_iterator_product::right_neg_` [protected]

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

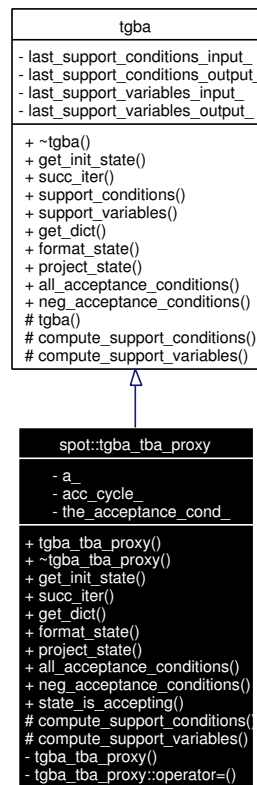
- [tgba/tgbaproduct.hh](#)

## 7.81 spot::tgba\_tba\_proxy Class Reference

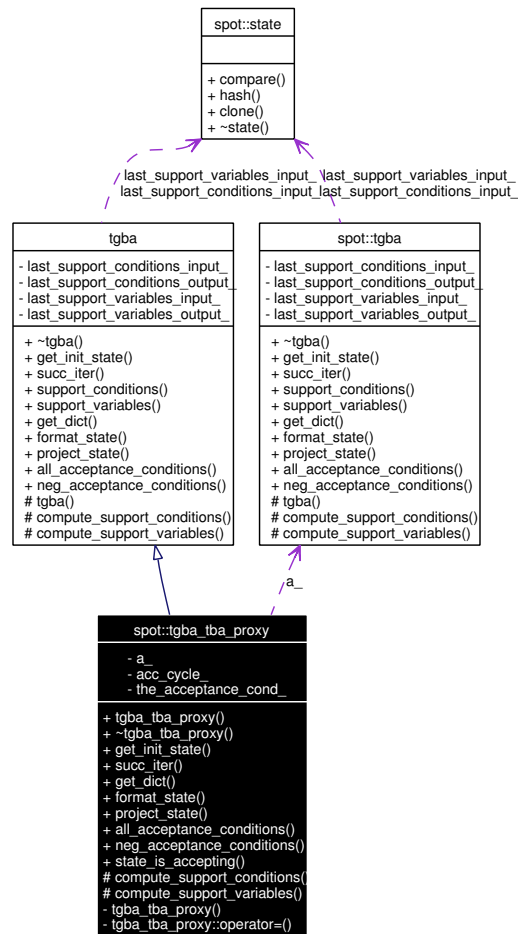
Degeneralize a [spot::tgba](#) on the fly.

```
#include <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 automata.*
- 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.*
- bool `state_is_accepting` (const `state *state`) const  
*Whether the state is accepting.*
- 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.*

### 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_tba_proxy` (const `tgba_tba_proxy &`)
- `tgba_tba_proxy & tgba_tba_proxy::operator=` (const `tgba_tba_proxy &`)

### Private Attributes

- const `tgba * a_`
- `cycle_list acc_cycle_`
- bdd `the_acceptance_cond_`

#### 7.81.1 Detailed Description

Degeneralize a `spot::tgba` on the fly.

This class acts as a proxy in front of a `spot::tgba`, that should be degeneralized on the fly.

This automaton is a `spot::tgba`, but it will always have exactly one acceptance condition.

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  $\max(N,1)+1$  times more states and transitions.

## 7.81.2 Member Typedef Documentation

7.81.2.1 `typedef std::list<bdd> spot::tgba_tba_proxy::cycle_list`

## 7.81.3 Constructor & Destructor Documentation

7.81.3.1 `spot::tgba_tba_proxy::tgba_tba_proxy (const tgba * a)`

7.81.3.2 `virtual spot::tgba_tba_proxy::~~tgba_tba_proxy ()` [virtual]

7.81.3.3 `spot::tgba_tba_proxy::tgba_tba_proxy (const tgba_tba_proxy &)` [private]

## 7.81.4 Member Function Documentation

7.81.4.1 `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`.

7.81.4.2 `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`.

7.81.4.3 `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`.

7.81.4.4 `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`.

7.81.4.5 `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](#).

#### 7.81.4.6 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](#).

#### 7.81.4.7 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](#).

#### 7.81.4.8 virtual [state\\*](#) spot::tgba\_tba\_proxy::project\_state (const [state](#) \* *s*, const [tgba](#) \* *t*) const [virtual]

Project a state on an automata.

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](#).

#### 7.81.4.9 bool spot::tgba\_tba\_proxy::state\_is\_accepting (const [state](#) \* *state*) const

Whether the state is accepting.

A particularity of a `spot::tgba_tba_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 to many algorithms working on degeneralized automata with state acceptance conditions.

#### 7.81.4.10 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](#).

#### 7.81.4.11 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.

#### 7.81.4.12 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.

#### 7.81.4.13 [tgba\\_tba\\_proxy&](#) spot::tgba\_tba\_proxy::tgba\_tba\_proxy::operator= (const [tgba\\_tba\\_proxy](#) &) [private]

### 7.81.5 Member Data Documentation

#### 7.81.5.1 const [tgba](#)\* spot::tgba\_tba\_proxy::a\_ [private]

#### 7.81.5.2 [cycle\\_list](#) spot::tgba\_tba\_proxy::acc\_cycle\_ [private]

#### 7.81.5.3 bdd spot::tgba\_tba\_proxy::the\_acceptance\_cond\_ [private]

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

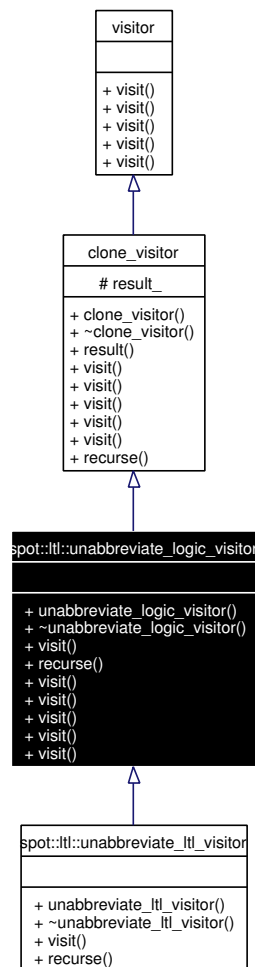
- [tgba/tgbatba.hh](#)

## 7.82 spot::ltl::unabbreviate\_logic\_visitor Class Reference

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

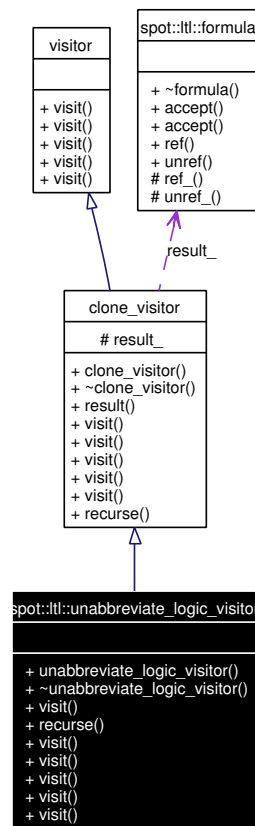
```
#include <lunabbrev.hh>
```

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)`
- `void visit(atomic_prop *ap)`
- `void visit(unop *uo)`
- `void visit(binop *bo)`
- `void visit(multop *mo)`
- `void visit(constant *c)`
- `formula * result()` const

### Protected Attributes

- `formula * result_`

### Private Types

- `typedef clone_visitor super`

### 7.82.1 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.

### 7.82.2 Member Typedef Documentation

**7.82.2.1** typedef [clone\\_visitor](#) [spot::ltl::unabbreviate\\_logic\\_visitor::super](#) [private]

Reimplemented in [spot::ltl::unabbreviate\\_ltl\\_visitor](#).

### 7.82.3 Constructor & Destructor Documentation

**7.82.3.1** [spot::ltl::unabbreviate\\_logic\\_visitor::unabbreviate\\_logic\\_visitor](#) ()

**7.82.3.2** virtual [spot::ltl::unabbreviate\\_logic\\_visitor::~~unabbreviate\\_logic\\_visitor](#) ()  
[virtual]

### 7.82.4 Member Function Documentation

**7.82.4.1** virtual [formula\\*](#) [spot::ltl::unabbreviate\\_logic\\_visitor::recurse](#) ([formula](#) \**f*) [virtual]

Reimplemented from [spot::ltl::clone\\_visitor](#).

Reimplemented in [spot::ltl::unabbreviate\\_ltl\\_visitor](#).

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

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

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

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

**7.82.4.6** void [spot::ltl::clone\\_visitor::visit](#) ([unop](#) \**uo*)

**7.82.4.7** void [spot::ltl::clone\\_visitor::visit](#) ([atomic\\_prop](#) \**ap*)

**7.82.4.8** void [spot::ltl::unabbreviate\\_logic\\_visitor::visit](#) ([binop](#) \**bo*) [virtual]

Reimplemented from [spot::ltl::clone\\_visitor](#).

### 7.82.5 Member Data Documentation

#### 7.82.5.1 formula\* spot::ltl::clone\_visitor::result\_ [protected, inherited]

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

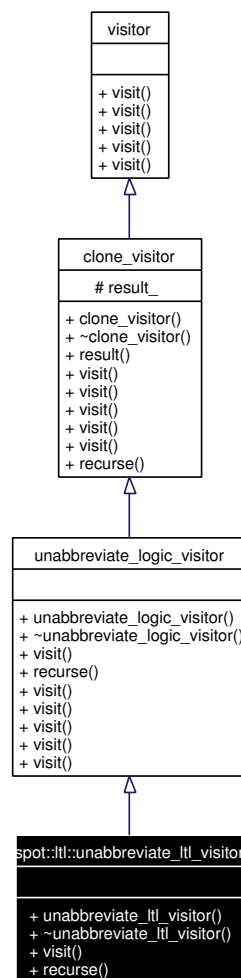
- [ltlvisit/unabbrev.hh](#)

## 7.83 spot::ltl::unabbreviate\_ltl\_visitor Class Reference

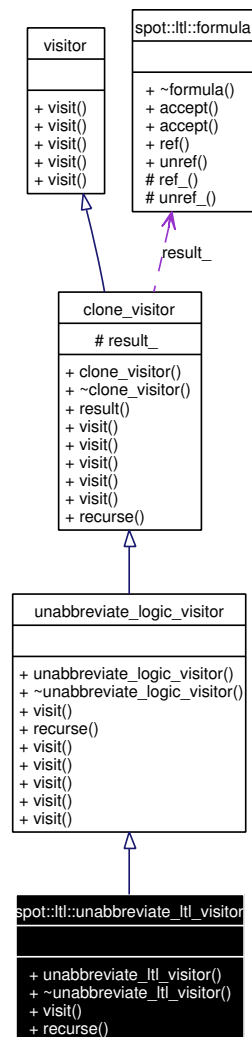
Clone and rewrite a formula to remove most of the abbreviated LTL and logical operators.

```
#include <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

### 7.83.1 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.

### 7.83.2 Member Typedef Documentation

**7.83.2.1** typedef [unabbreviate\\_logic\\_visitor](#) [spot::ltl::unabbreviate\\_ltl\\_visitor::super](#) [private]

Reimplemented from [spot::ltl::unabbreviate\\_logic\\_visitor](#).

### 7.83.3 Constructor & Destructor Documentation

**7.83.3.1** [spot::ltl::unabbreviate\\_ltl\\_visitor::unabbreviate\\_ltl\\_visitor \(\)](#)

**7.83.3.2** virtual [spot::ltl::unabbreviate\\_ltl\\_visitor::~~unabbreviate\\_ltl\\_visitor \(\)](#) [virtual]

### 7.83.4 Member Function Documentation

**7.83.4.1** [formula\\*](#) [spot::ltl::unabbreviate\\_ltl\\_visitor::recurse \(formula \\*f\)](#) [virtual]

Reimplemented from [spot::ltl::unabbreviate\\_logic\\_visitor](#).

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

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

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

**7.83.4.5** void [spot::ltl::clone\\_visitor::visit \(atomic\\_prop \\*ap\)](#) [inherited]

**7.83.4.6** void [spot::ltl::unabbreviate\\_logic\\_visitor::visit \(binop \\*bo\)](#) [virtual, inherited]

Reimplemented from [spot::ltl::clone\\_visitor](#).

**7.83.4.7** void spot::ltl::unabbreviate\_ltl\_visitor::visit (**unop** \* *uo*) [virtual]

Reimplemented from [spot::ltl::clone\\_visitor](#).

### 7.83.5 Member Data Documentation

**7.83.5.1** formula\* [spot::ltl::clone\\_visitor::result\\_](#) [protected, inherited]

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

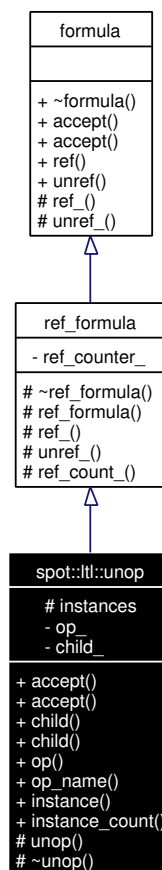
- ltlvisit/[tunabbrev.hh](#)

## 7.84 spot::ltl::unop Class Reference

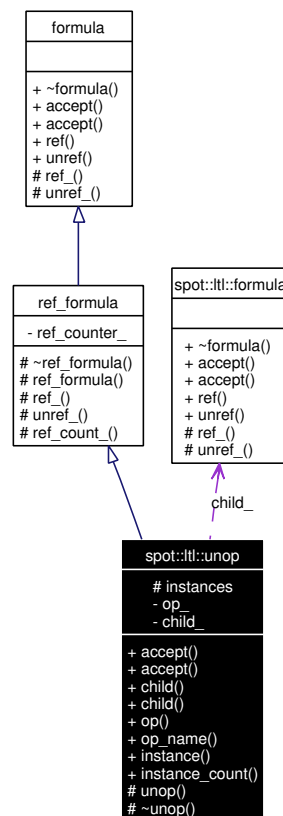
Unary operator.

```
#include <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*

### Static Public Member Functions

- [unop \\* instance \(type op, formula \\*child\)](#)
- unsigned [instance\\_count \(\)](#)  
*Number of instantiated unary operators. For debugging.*
- 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.*

### Static Protected Attributes

- [map instances](#)

### Private Attributes

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

#### 7.84.1 Detailed Description

Unary operator.



### 7.84.2 Member Typedef Documentation

7.84.2.1 typedef std::map<pair, formula\*> spot::ltl::unop::map [protected]

7.84.2.2 typedef std::pair<type, formula\*> spot::ltl::unop::pair [protected]

### 7.84.3 Member Enumeration Documentation

7.84.3.1 enum spot::ltl::unop::type

Enumeration values:

*Not*

*X*

*F*

*G*

### 7.84.4 Constructor & Destructor Documentation

7.84.4.1 spot::ltl::unop::unop (type op, formula \* child) [protected]

7.84.4.2 virtual spot::ltl::unop::~~unop () [protected, virtual]

### 7.84.5 Member Function Documentation

7.84.5.1 virtual void spot::ltl::unop::accept (const\_visitor & v) const [virtual]

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

Implements spot::ltl::formula.

7.84.5.2 virtual void spot::ltl::unop::accept (visitor & v) [virtual]

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

Implements spot::ltl::formula.

7.84.5.3 formula\* spot::ltl::unop::child ()

Get the sole operand of this operator.

7.84.5.4 const formula\* spot::ltl::unop::child () const

Get the sole operand of this operator.

7.84.5.5 unop\* spot::ltl::unop::instance (type op, formula \* child) [static]

Build an unary operator with operation *op* and child *child*.

7.84.5.6 unsigned spot::ltl::unop::instance\_count () [static]

Number of instantiated unary operators. For debugging.

**7.84.5.7** `type spot::ltl::unop::op () const`

Get the type of this operator.

**7.84.5.8** `const char* spot::ltl::unop::op_name () const`

Get the type of this operator, as a string.

**7.84.5.9** `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.

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

increment reference counter if any

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

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

Number of references to this formula.

**7.84.5.12** `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.

**7.84.5.13** `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`.

**7.84.6** Member Data Documentation**7.84.6.1** `formula* spot::ltl::unop::child_ [private]`**7.84.6.2** `map spot::ltl::unop::instances [static, protected]`**7.84.6.3** `type spot::ltl::unop::op_ [private]`

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

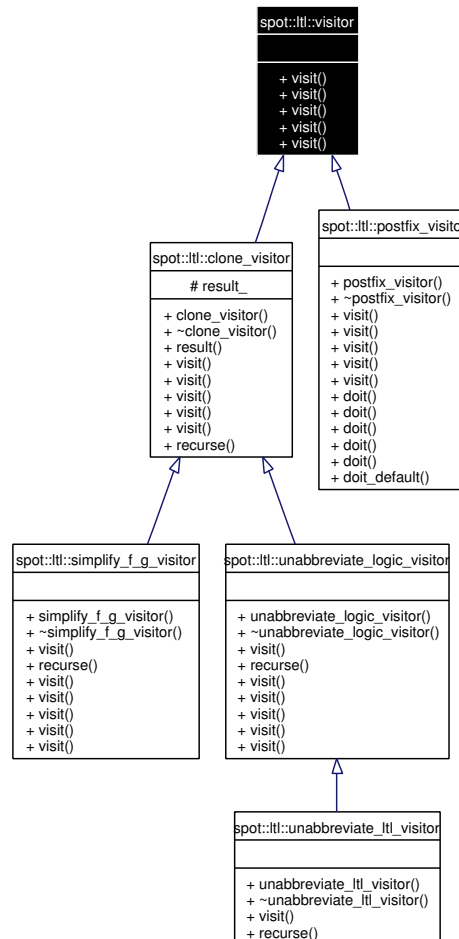
- `ltlast/unop.hh`

## 7.85 spot::ltl::visitor Struct Reference

Formula visitor that can modify the formula.

```
#include <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

#### 7.85.1 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.

## 7.85.2 Member Function Documentation

**7.85.2.1** `virtual void spot::ltl::visitor::visit (multop * node)` [pure virtual]

Implemented in `spot::ltl::clone_visitor`, and `spot::ltl::postfix_visitor`.

**7.85.2.2** `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`.

**7.85.2.3** `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`.

**7.85.2.4** `virtual void spot::ltl::visitor::visit (constant * node)` [pure virtual]

Implemented in `spot::ltl::clone_visitor`, and `spot::ltl::postfix_visitor`.

**7.85.2.5** `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`

## 8 spot File Documentation

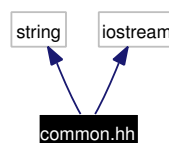
### 8.1 /home/adl/proj/spot/doc/mainpage.dox File Reference

### 8.2 gspn/common.hh File Reference

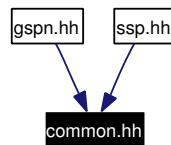
```
#include <string>
```

```
#include <iostream>
```

Include dependency graph for `common.hh`:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace `spot`

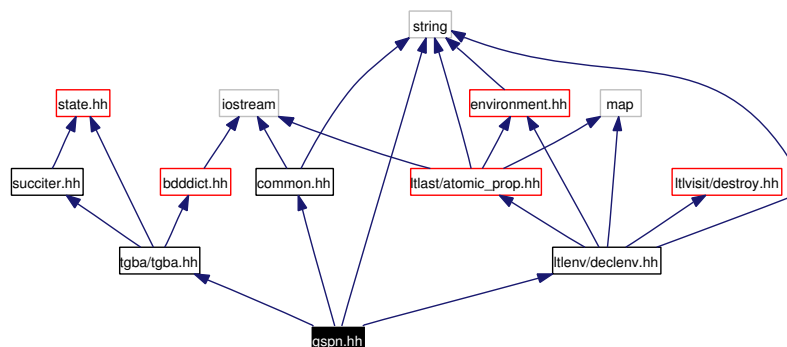
## 8.3 gspn/gspn.hh File Reference

```

#include <string>
#include "tgba/tgba.hh"
#include "common.hh"
#include "ltlenv/declenv.hh"

```

Include dependency graph for gspn.hh:



### Namespaces

- namespace `spot`

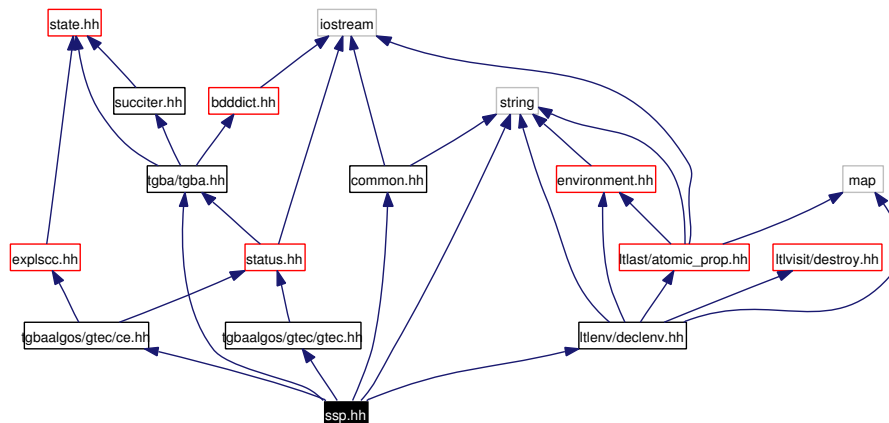
## 8.4 gspn/ssp.hh File Reference

```

#include <string>
#include "tgba/tgba.hh"
#include "common.hh"
#include "tgbaalgos/gtec/gtec.hh"
#include "tgbaalgos/gtec/ce.hh"
#include "ltlenv/declenv.hh"

```

Include dependency graph for ssp.hh:



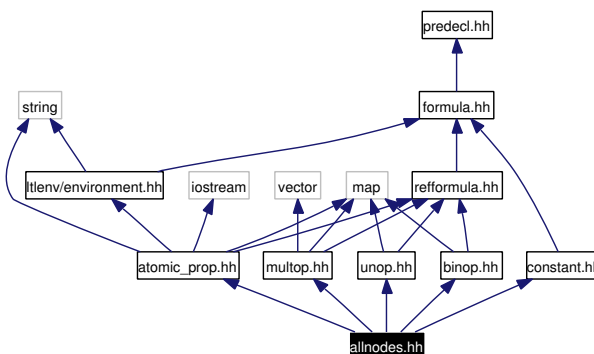
## Namespaces

- namespace **spot**

## 8.5 Itlast/allnodes.hh File Reference

```
#include "binop.hh"
#include "unop.hh"
#include "multop.hh"
#include "atomic_prop.hh"
#include "constant.hh"
```

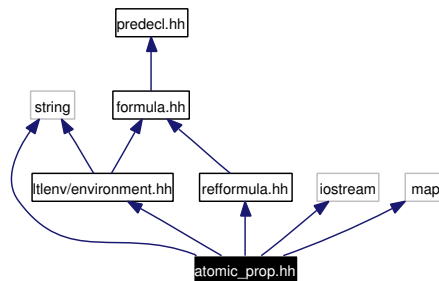
Include dependency graph for allnodes.hh:



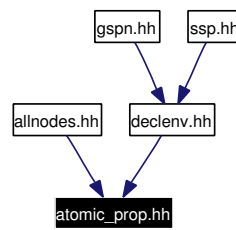
## 8.6 Itlast/atomic\_prop.hh File Reference

```
#include <string>
#include <iostream>
#include <map>
```

```
#include "reformula.hh"
#include "ltlenv/environment.hh"
Include dependency graph for atomic_prop.hh:
```



This graph shows which files directly or indirectly include this file:

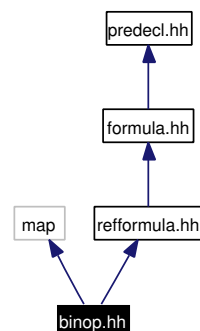


### Namespaces

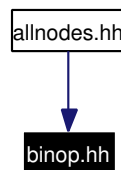
- namespace `spot`
- namespace `spot::ltl`

## 8.7 Itlast/binop.hh File Reference

```
#include <map>
#include "reformula.hh"
Include dependency graph for binop.hh:
```



This graph shows which files directly or indirectly include this file:



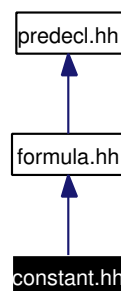
#### Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

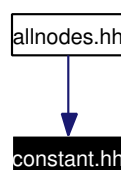
## 8.8 Itlast/constant.hh File Reference

```
#include "formula.hh"
```

Include dependency graph for constant.hh:



This graph shows which files directly or indirectly include this file:



#### Namespaces

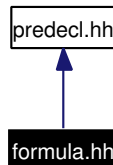
- namespace [spot](#)
- namespace [spot::ltl](#)



## 8.9 Itlast/formula.hh File Reference

```
#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:~l](#)

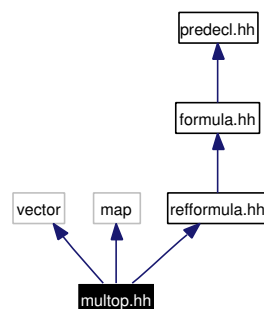
## 8.10 Itlast/multop.hh File Reference

```
#include <vector>
```

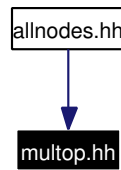
```
#include <map>
```

```
#include "refformula.hh"
```

Include dependency graph for multop.hh:



This graph shows which files directly or indirectly include this file:

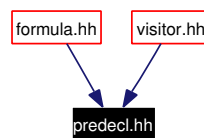


### Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

## 8.11 Itlast/predecl.hh File Reference

This graph shows which files directly or indirectly include this file:



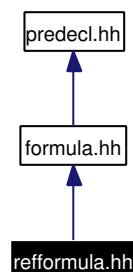
### Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

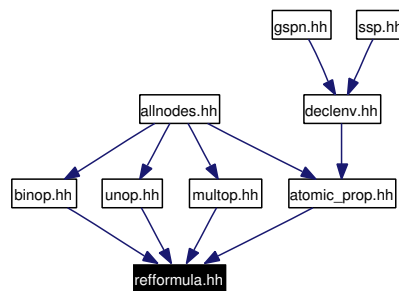
## 8.12 Itlast/reformula.hh File Reference

```
#include "formula.hh"
```

Include dependency graph for reformula.hh:



This graph shows which files directly or indirectly include this file:



### Namespaces

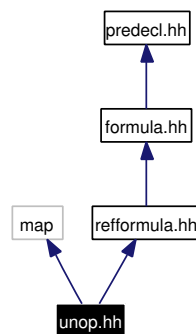
- namespace `spot`
- namespace `spot::ltl`

## 8.13 Itlast/unop.hh File Reference

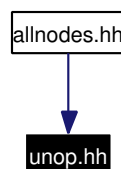
```
#include <map>
```

```
#include "refformula.hh"
```

Include dependency graph for `unop.hh`:



This graph shows which files directly or indirectly include this file:



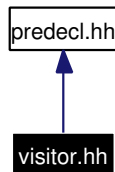
### Namespaces

- namespace `spot`
- namespace `spot::ltl`

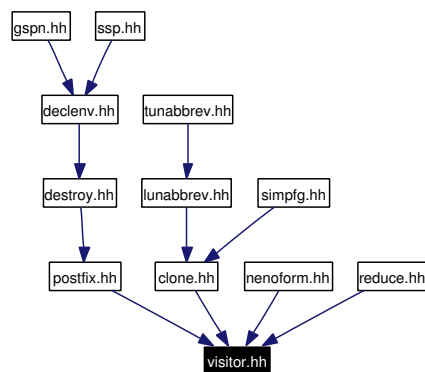
## 8.14 ltlast/visitor.hh File Reference

```
#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](#)

## 8.15 ltlenv/declenv.hh File Reference

```
#include "environment.hh"
```

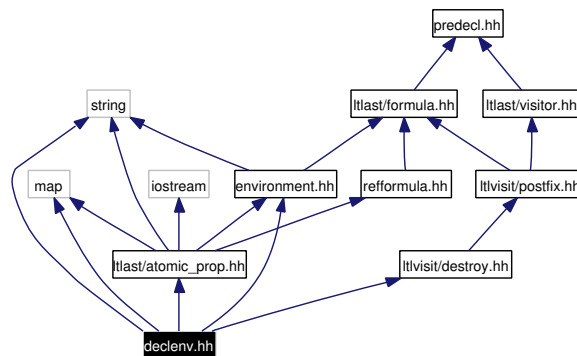
```
#include <string>
```

```
#include <map>
```

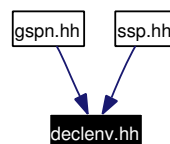
```
#include "ltlvisit/destroy.hh"
```

```
#include "ltlast/atomic_prop.hh"
```

Include dependency graph for declenv.hh:



This graph shows which files directly or indirectly include this file:



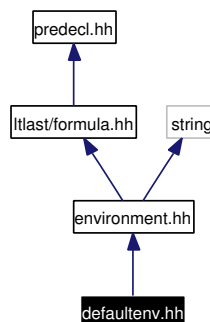
## Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

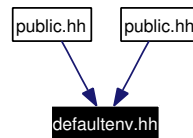
## 8.16 Itlenv/defaultenv.hh File Reference

```
#include "environment.hh"
```

Include dependency graph for defaultenv.hh:



This graph shows which files directly or indirectly include this file:



### Namespaces

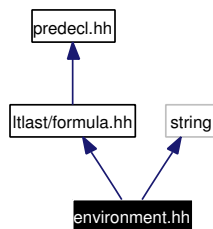
- namespace [spot](#)
- namespace [spot::ltl](#)

## 8.17 Itlenv/environment.hh File Reference

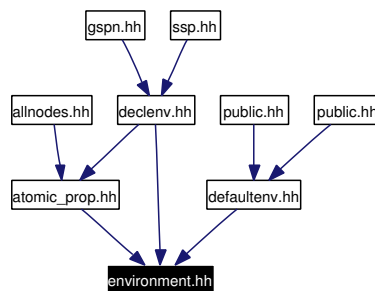
```
#include "ltlast/formula.hh"
```

```
#include <string>
```

Include dependency graph for environment.hh:



This graph shows which files directly or indirectly include this file:



### Namespaces

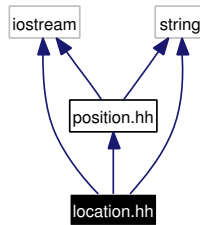
- namespace [spot](#)
- namespace [spot::ltl](#)

## 8.18 Itlparse/location.hh File Reference

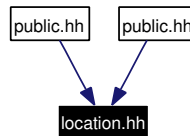
```
#include <iostream>
```

```
#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`

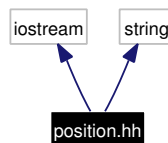
### 8.18.1 Detailed Description

Define the Location class.

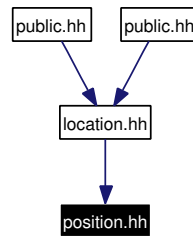
## 8.19 Itlparse/position.hh File Reference

```
#include <iostream>
#include <string>
```

Include dependency graph for position.hh:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [yy](#)

### 8.19.1 Detailed Description

Define the Location class.

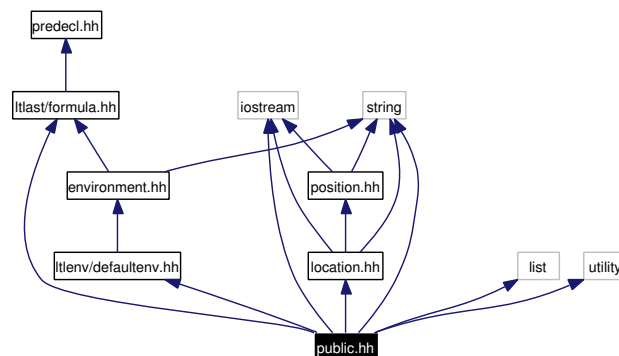
## 8.20 Itlparse/public.hh File Reference

```

#include "ltlast/formula.hh"
#include "location.hh"
#include "ltlenv/defaultenv.hh"
#include <string>
#include <list>
#include <utility>
#include <iostream>

```

Include dependency graph for public.hh:



## Namespaces

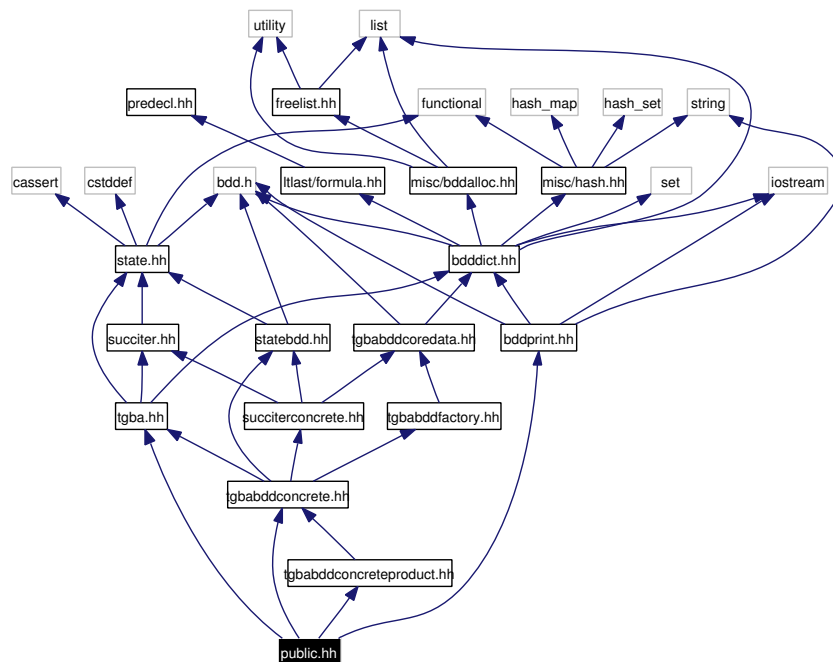
- namespace [spot](#)
- namespace [spot::ltl](#)



## 8.21 tgba/public.hh File Reference

```
#include "tgba.hh"
#include "tgbabddconcrete.hh"
#include "tgbabddconcretetproduct.hh"
#include "bddprint.hh"
```

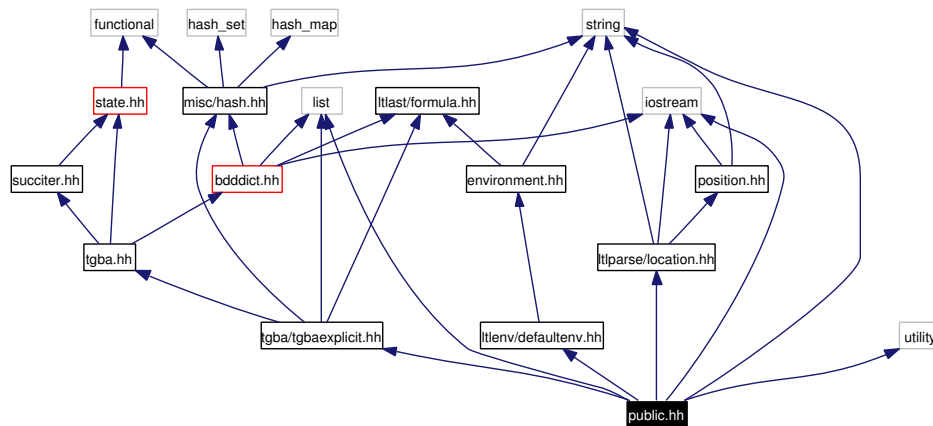
Include dependency graph for public.hh:



## 8.22 tgbaparse/public.hh File Reference

```
#include "tgba/tgbaexplicit.hh"
#include "ltlparse/location.hh"
#include "ltlenv/defaultenv.hh"
#include <string>
#include <list>
#include <utility>
#include <iostream>
```

Include dependency graph for public.hh:



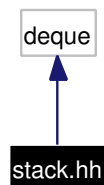
### Namespaces

- namespace [spot](#)

## 8.23 Itlparse/stack.hh File Reference

```
#include <deque>
```

Include dependency graph for stack.hh:



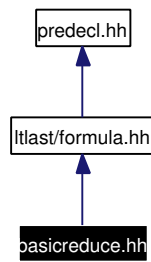
### Namespaces

- namespace [yy](#)

## 8.24 Itlvisit/basicreduce.hh File Reference

```
#include "ltlast/formula.hh"
```

Include dependency graph for basicreduce.hh:



### Namespaces

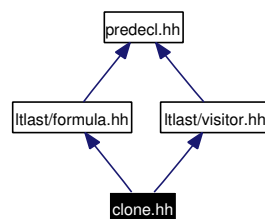
- namespace [spot](#)
- namespace [spot::ltl](#)

## 8.25 Itlvisit/clone.hh File Reference

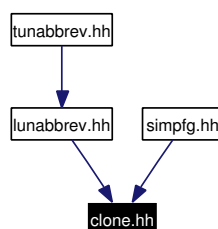
```
#include "ltlast/formula.hh"
```

```
#include "ltlast/visitor.hh"
```

Include dependency graph for clone.hh:



This graph shows which files directly or indirectly include this file:



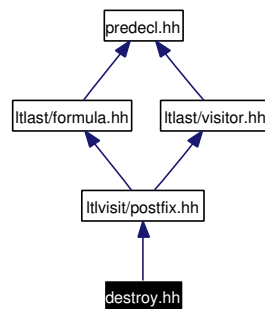
### Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

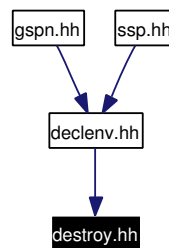
## 8.26 ltlvisit/destroy.hh File Reference

```
#include "ltlvisit/postfix.hh"
```

Include dependency graph for destroy.hh:



This graph shows which files directly or indirectly include this file:



### Namespaces

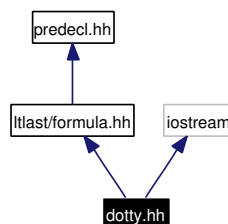
- namespace `spot`
- namespace `spot::ltl`

## 8.27 ltlvisit/dotty.hh File Reference

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

```
#include <iostream>
```

Include dependency graph for dotty.hh:



## Namespaces

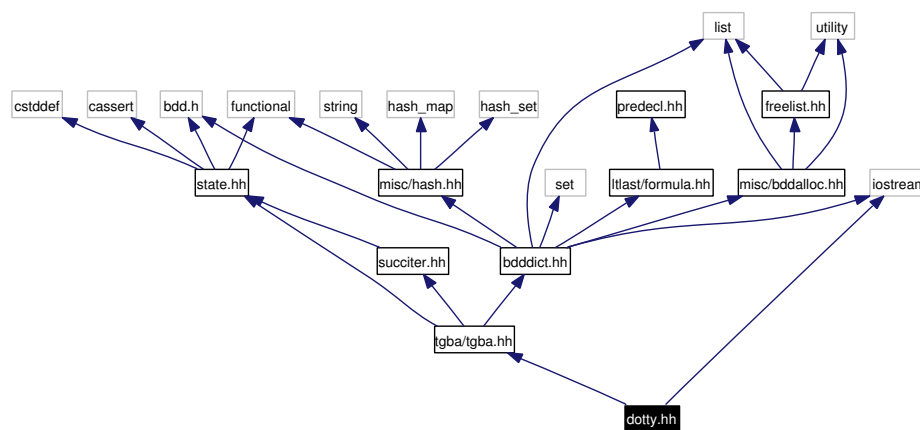
- namespace [spot](#)
- namespace [spot::ltl](#)

## 8.28 tgbalgorithms/dotty.hh File Reference

```
#include "tgb/tgba.hh"
```

```
#include <iostream>
```

Include dependency graph for dotty.hh:



## Namespaces

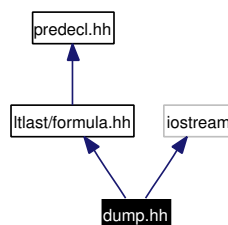
- namespace [spot](#)

## 8.29 ltlvisit/dump.hh File Reference

```
#include "ltlast/formula.hh"
```

```
#include <iostream>
```

Include dependency graph for dump.hh:



## Namespaces

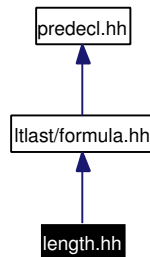
- namespace [spot](#)

- namespace [spot::ltl](#)

### 8.30 Itlvisit/length.hh File Reference

```
#include "ltlast/formula.hh"
```

Include dependency graph for length.hh:



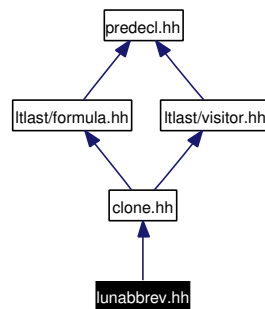
#### Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

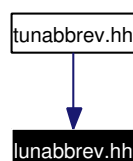
### 8.31 Itlvisit/lunabbrev.hh File Reference

```
#include "clone.hh"
```

Include dependency graph for lunabbrev.hh:



This graph shows which files directly or indirectly include this file:



### Namespaces

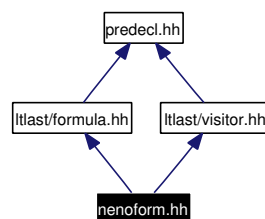
- namespace [spot](#)
- namespace [spot::ltl](#)

## 8.32 Itlvisit/nenoform.hh File Reference

```
#include "ltlast/formula.hh"
```

```
#include "ltlast/visitor.hh"
```

Include dependency graph for nenoform.hh:



### Namespaces

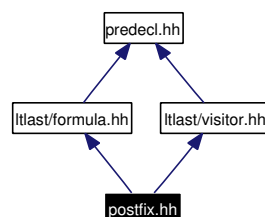
- namespace [spot](#)
- namespace [spot::ltl](#)

## 8.33 Itlvisit/postfix.hh File Reference

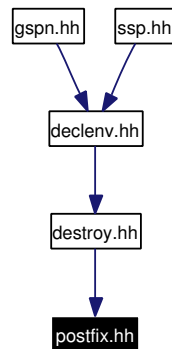
```
#include "ltlast/formula.hh"
```

```
#include "ltlast/visitor.hh"
```

Include dependency graph for postfix.hh:



This graph shows which files directly or indirectly include this file:



### Namespaces

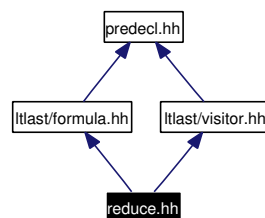
- namespace `spot`
- namespace `spot::ltl`

## 8.34 Itlvisit/reduce.hh File Reference

```
#include "ltlast/formula.hh"
```

```
#include "ltlast/visitor.hh"
```

Include dependency graph for reduce.hh:



### Namespaces

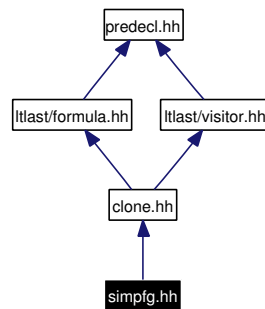
- namespace `spot`
- namespace `spot::ltl`

## 8.35 Itlvisit/simpfg.hh File Reference

```
#include "clone.hh"
```

Include dependency graph for simpfg.hh:





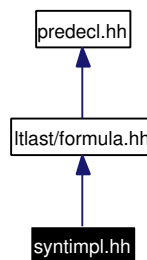
### Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

## 8.36 ltlvisit/syntimpl.hh File Reference

```
#include "ltlast/formula.hh"
```

Include dependency graph for syntimpl.hh:



### Namespaces

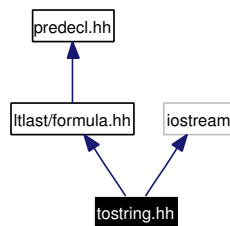
- namespace [spot](#)
- namespace [spot::ltl](#)

## 8.37 ltlvisit/tostring.hh File Reference

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

```
#include <iostream>
```

Include dependency graph for tostring.hh:



### Namespaces

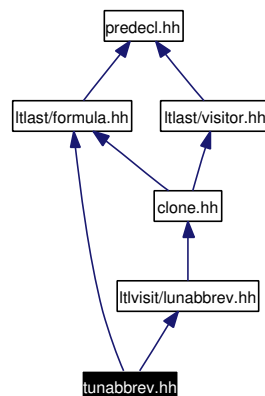
- namespace `spot`
- namespace `spot::ltl`

## 8.38 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`

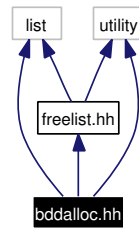
## 8.39 misc/bddalloc.hh File Reference

```
#include "freelist.hh"
```

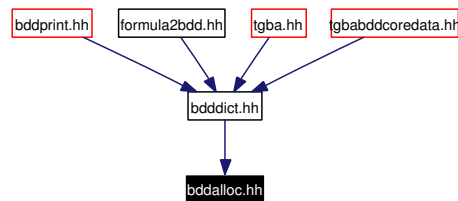
```
#include <list>
```

```
#include <utility>
```

Include dependency graph for `bddalloc.hh`:



This graph shows which files directly or indirectly include this file:



## Namespaces

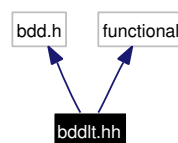
- namespace `spot`

## 8.40 misc/bddlt.hh File Reference

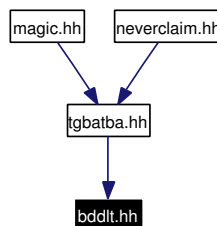
```
#include <bdd.h>
```

```
#include <functional>
```

Include dependency graph for `bddlt.hh`:



This graph shows which files directly or indirectly include this file:



### Namespaces

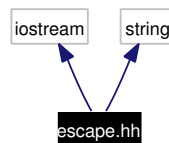
- namespace [spot](#)

## 8.41 misc/escape.hh File Reference

```
#include <iostream>
```

```
#include <string>
```

Include dependency graph for escape.hh:



### Namespaces

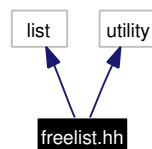
- namespace [spot](#)

## 8.42 misc/freelist.hh File Reference

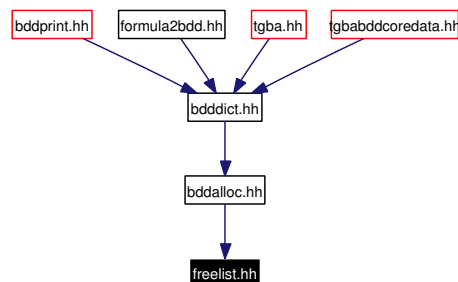
```
#include <list>
```

```
#include <utility>
```

Include dependency graph for freelist.hh:



This graph shows which files directly or indirectly include this file:



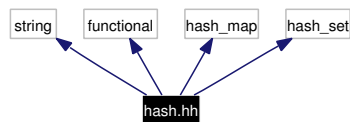
**Namespaces**

- namespace [spot](#)

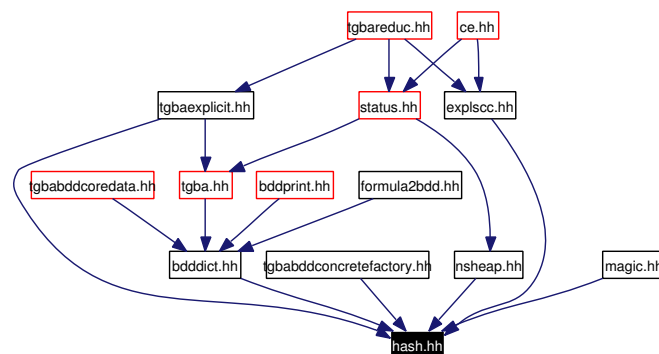
**8.43 misc/hash.hh File Reference**

```
#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:

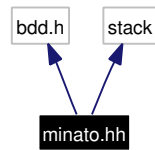
**Namespaces**

- namespace [spot](#)

**8.44 misc/minato.hh File Reference**

```
#include <bdd.h>
#include <stack>
```

Include dependency graph for minato.hh:



### Namespaces

- namespace [spot](#)

## 8.45 misc/version.hh File Reference

### Namespaces

- namespace [spot](#)

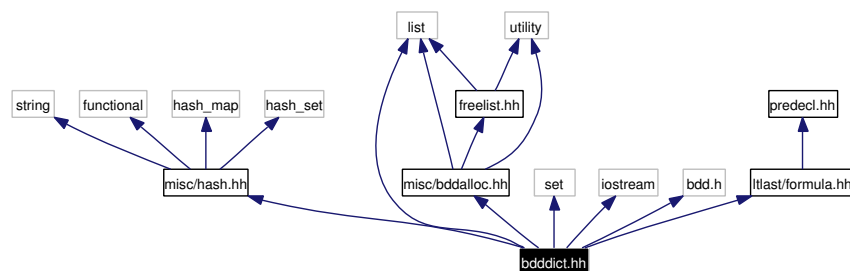
## 8.46 tgba/bdddict.hh File Reference

```

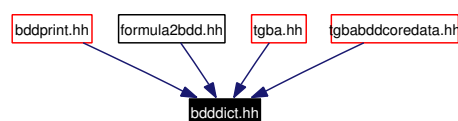
#include "misc/hash.hh"
#include <list>
#include <set>
#include <iostream>
#include <bdd.h>
#include "ltlast/formula.hh"
#include "misc/bddalloc.hh"

```

Include dependency graph for bdddict.hh:



This graph shows which files directly or indirectly include this file:



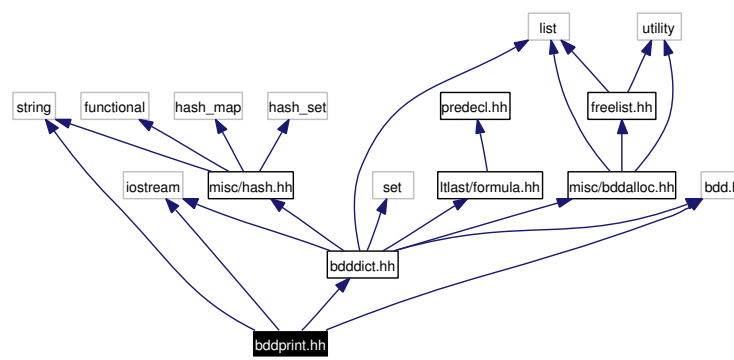
## Namespaces

- namespace [spot](#)

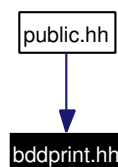
## 8.47 tgba/bddprint.hh File Reference

```
#include <string>
#include <iostream>
#include "bdddict.hh"
#include <bdd.h>
```

Include dependency graph for bddprint.hh:



This graph shows which files directly or indirectly include this file:



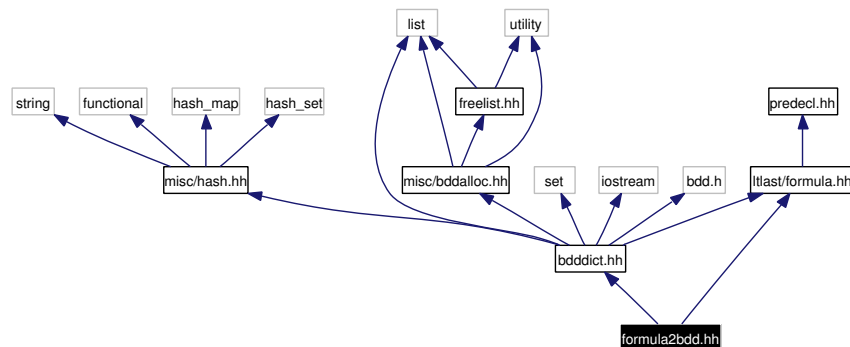
## Namespaces

- namespace [spot](#)

## 8.48 tgba/formula2bdd.hh File Reference

```
#include "bdddict.hh"
#include "ltlast/formula.hh"
```

Include dependency graph for formula2bdd.hh:



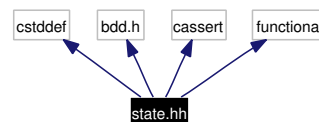
## Namespaces

- namespace [spot](#)

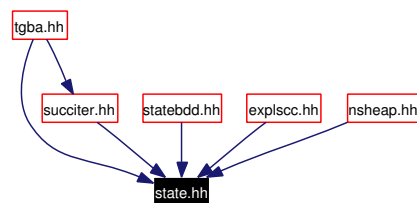
## 8.49 tgba/state.hh File Reference

```
#include <cstdint>
#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

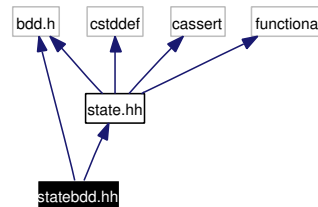
- namespace [spot](#)



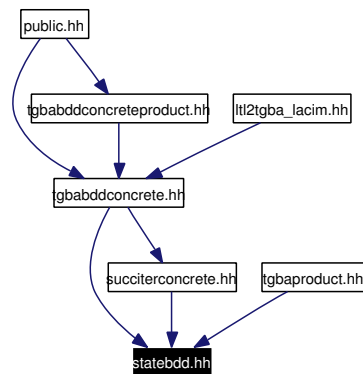
## 8.50 tgba/statebdd.hh File Reference

```
#include <bdd.h>
#include "state.hh"
```

Include dependency graph for statebdd.hh:



This graph shows which files directly or indirectly include this file:



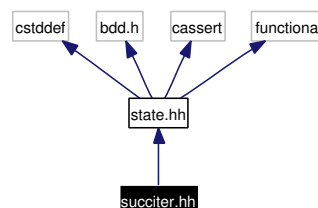
## Namespaces

- namespace **spot**

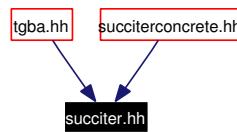
## 8.51 tgba/succiter.hh File Reference

```
#include "state.hh"
```

Include dependency graph for succiter.hh:



This graph shows which files directly or indirectly include this file:



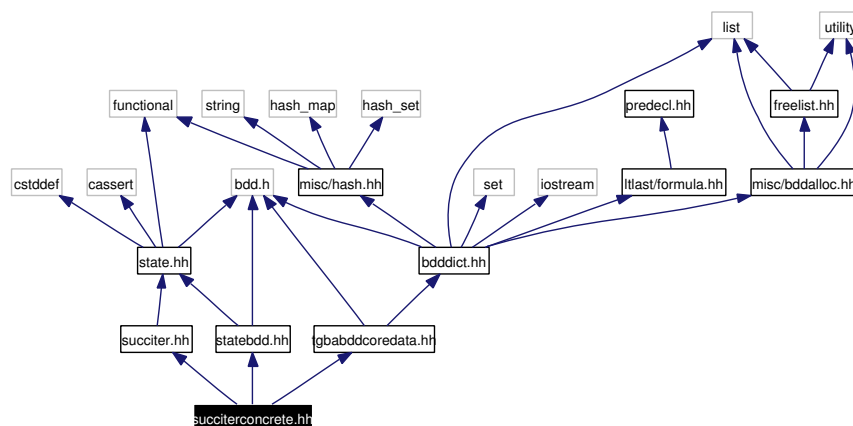
## Namespaces

- namespace [spot](#)

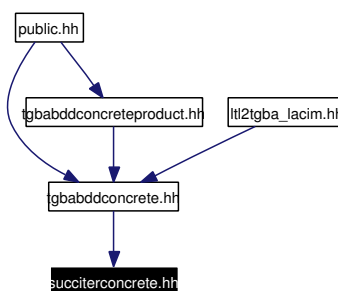
## 8.52 tgba/succiterconcrete.hh File Reference

```
#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:



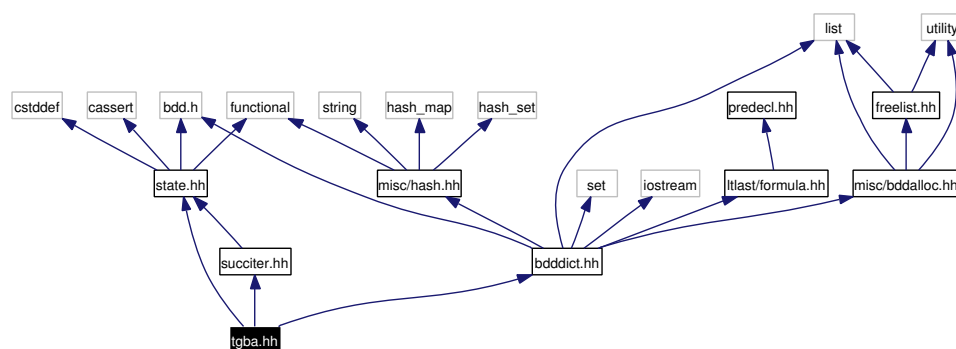
## Namespaces

- namespace [spot](#)

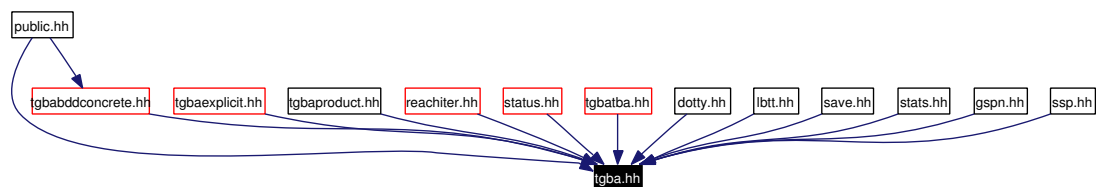
## 8.53 tgba/tgba.hh File Reference

```
#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:



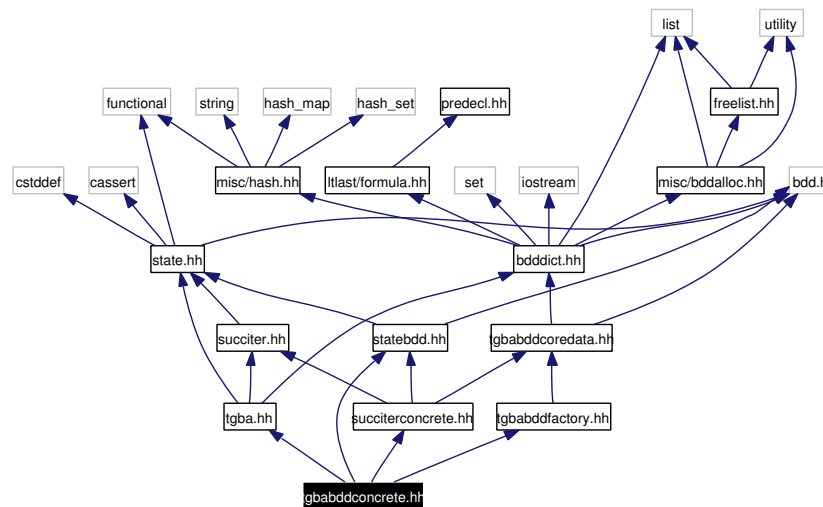
## Namespaces

- namespace [spot](#)

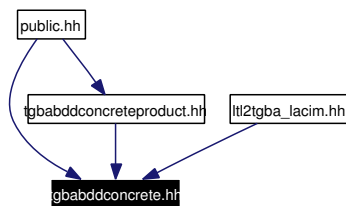
## 8.54 tgba/tgba\_bddconcrete.hh File Reference

```
#include "tgba.hh"
#include "statebdd.hh"
#include "tgba_bddfactory.hh"
#include "succiterconcrete.hh"
```

Include dependency graph for tgba\_bddconcrete.hh:



This graph shows which files directly or indirectly include this file:



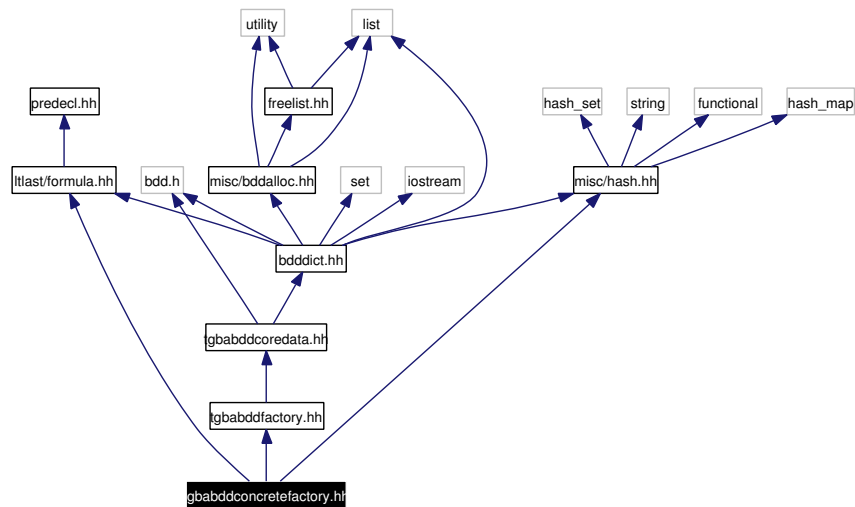
## Namespaces

- namespace **spot**

### 8.55 tgba/tgbabddconcretfactory.hh File Reference

```
#include "misc/hash.hh"
#include "ltlast/formula.hh"
#include "tgbabddfactory.hh"
```

Include dependency graph for tgbabddconcretefactory.hh:



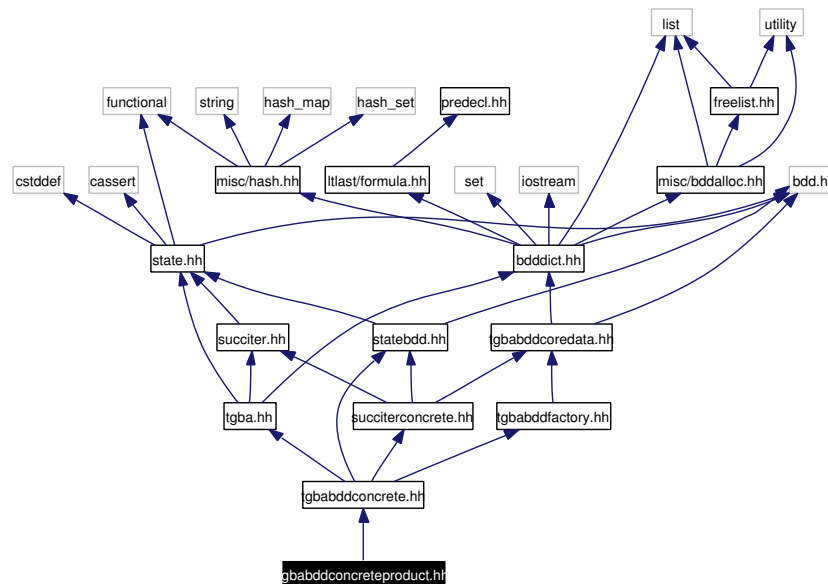
## Namespaces

- namespace [spot](#)

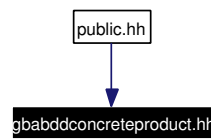
## 8.56 tgba/tgbabddconcreteproduct.hh File Reference

```
#include "tgbabddconcrete.hh"
```

Include dependency graph for tgbabddconcreteproduct.hh:



This graph shows which files directly or indirectly include this file:



## Namespaces

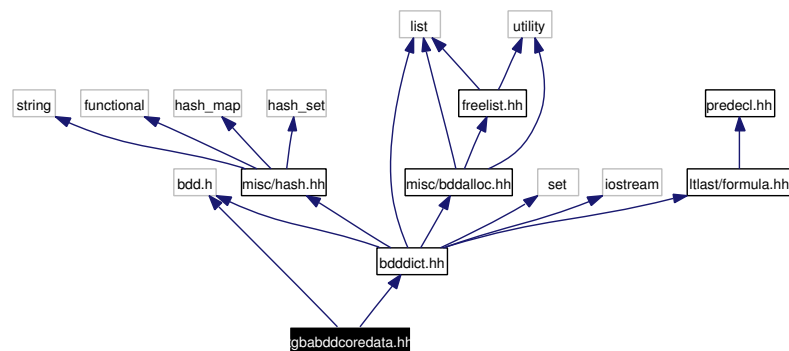
- namespace [spot](#)

## 8.57 tgba/tgbabddcoredata.hh File Reference

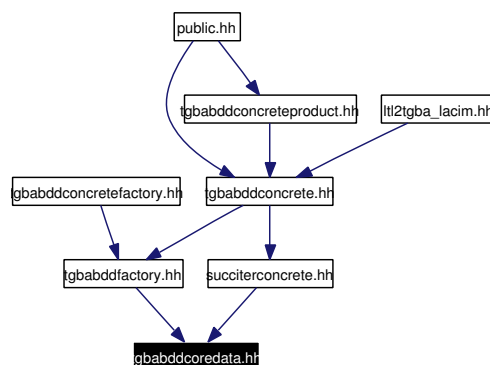
```
#include <bdd.h>
```

```
#include "bdddict.hh"
```

Include dependency graph for tgbabddcoredata.hh:



This graph shows which files directly or indirectly include this file:



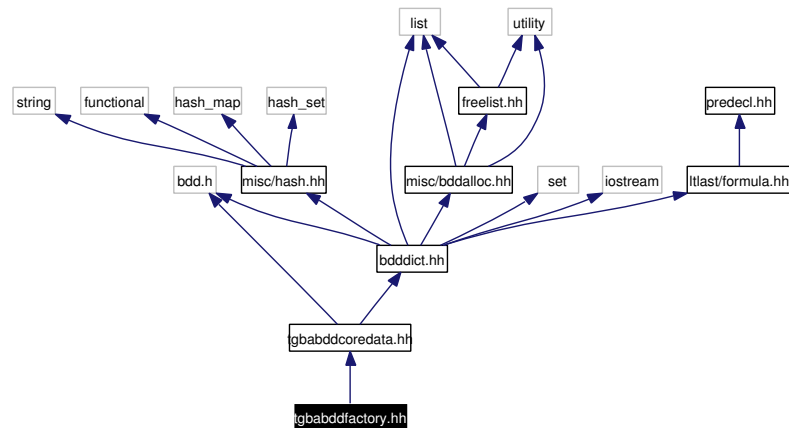
## Namespaces

- namespace [spot](#)

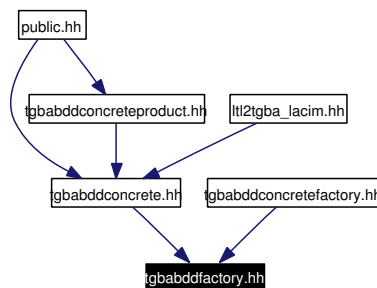
## 8.58 tgba/tgbabddfactory.hh File Reference

```
#include "tgbabddcoredata.hh"
```

Include dependency graph for tgbabddfactory.hh:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [spot](#)

## 8.59 tgba/tgbaexplicit.hh File Reference

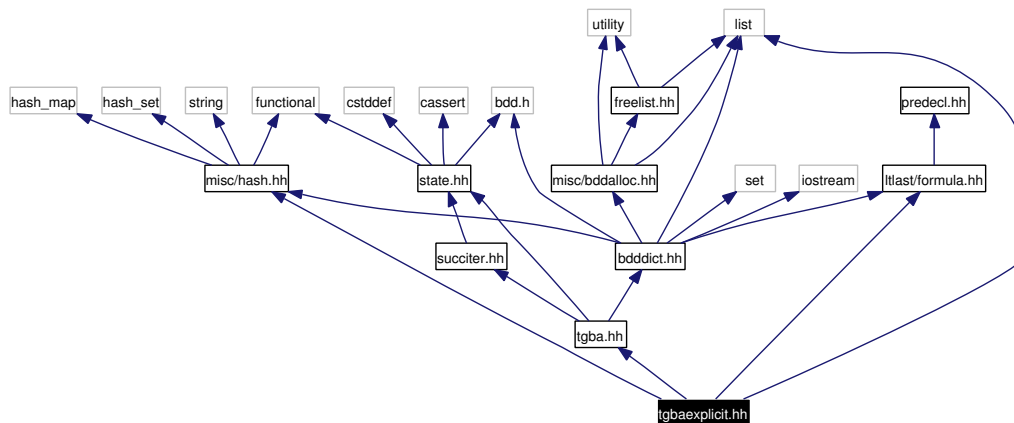
```
#include "misc/hash.hh"
```

```
#include <list>
```

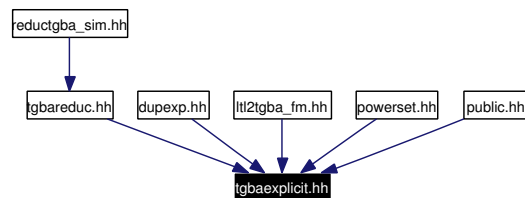
```
#include "tgba.hh"
```

```
#include "ltlast/formula.hh"
```

Include dependency graph for tgbaexplicit.hh:



This graph shows which files directly or indirectly include this file:



## Namespaces

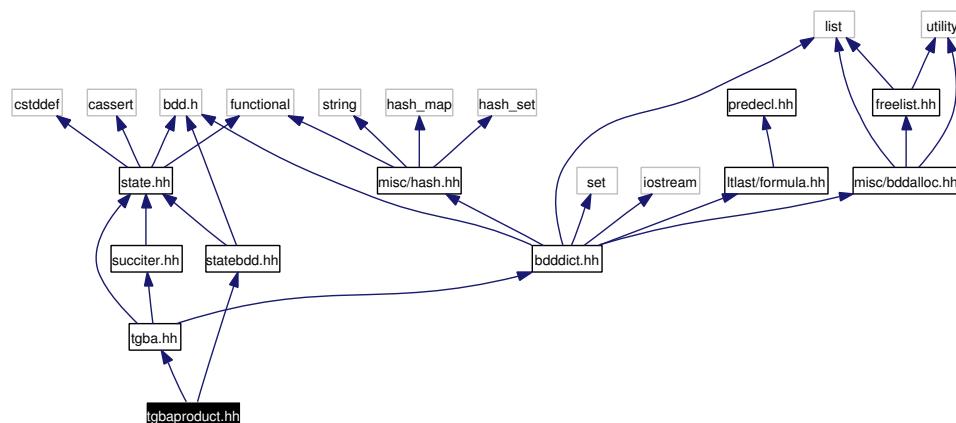
- namespace [spot](#)

## 8.60 tgba/tgbaproduct.hh File Reference

```
#include "tgba.hh"
```

```
#include "statebdd.hh"
```

Include dependency graph for tgbaproduct.hh:





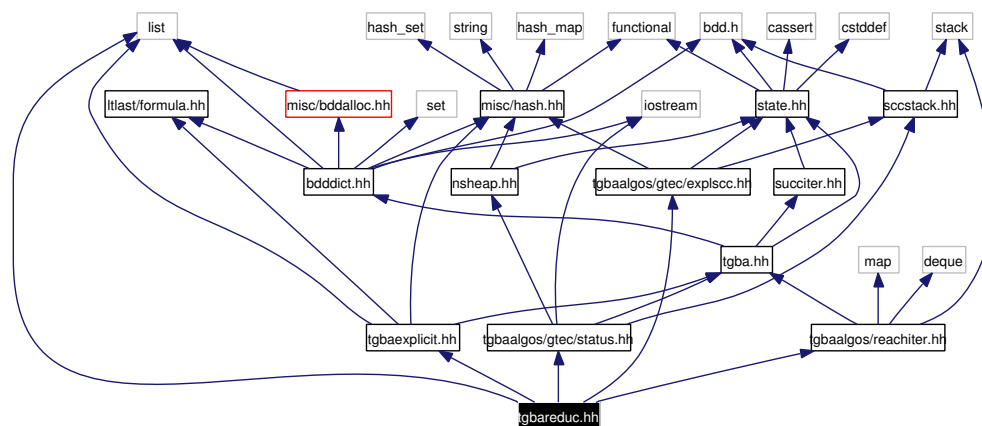
## Namespaces

- namespace **spot**

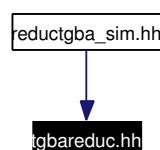
## 8.61 tgba/tgbareduc.hh File Reference

```
#include "tgbaexplicit.hh"
#include "tgbaalgos/reachiter.hh"
#include "tgbaalgos/gtec/explslcc.hh"
#include "tgbaalgos/gtec/status.hh"
#include <list>
```

Include dependency graph for `tgbareduc.hh`:



This graph shows which files directly or indirectly include this file:



## Namespaces

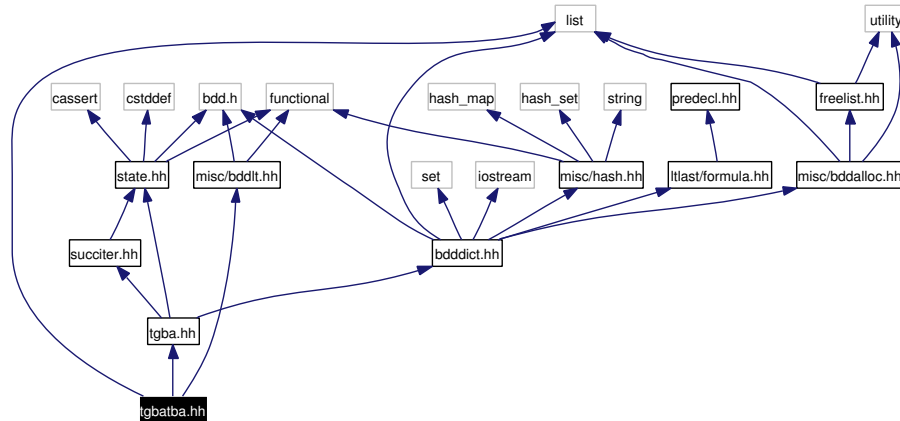
- namespace **spot**

## 8.62 tgba/tgbatba.hh File Reference

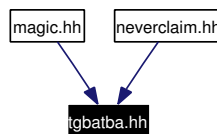
```
#include <list>
#include "tgba.hh"
```

```
#include "misc/bddlt.hh"
```

Include dependency graph for tgbatba.hh:



This graph shows which files directly or indirectly include this file:



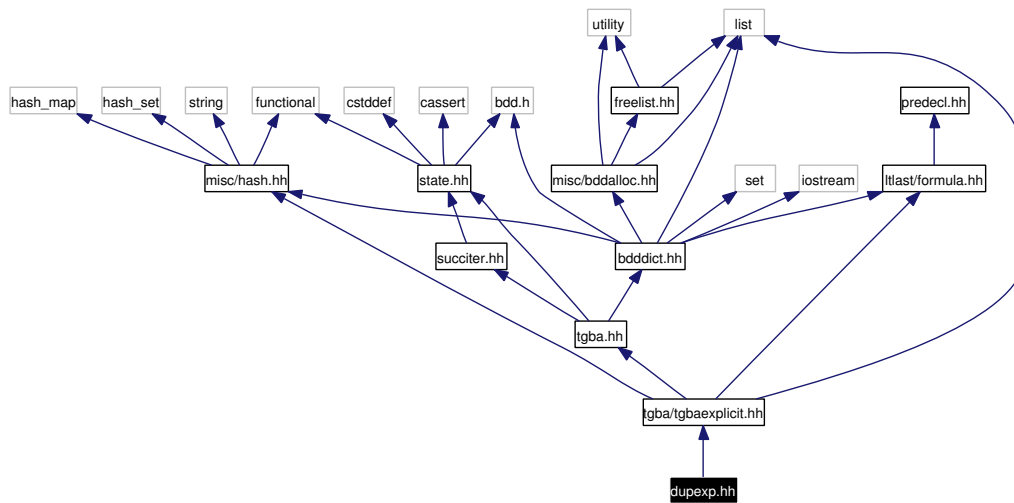
## Namespaces

- namespace [spot](#)

## 8.63 tgbaalgos/dupeexp.hh File Reference

```
#include "tgba/tgbaexplicit.hh"
```

Include dependency graph for dupeexp.hh:



## Namespaces

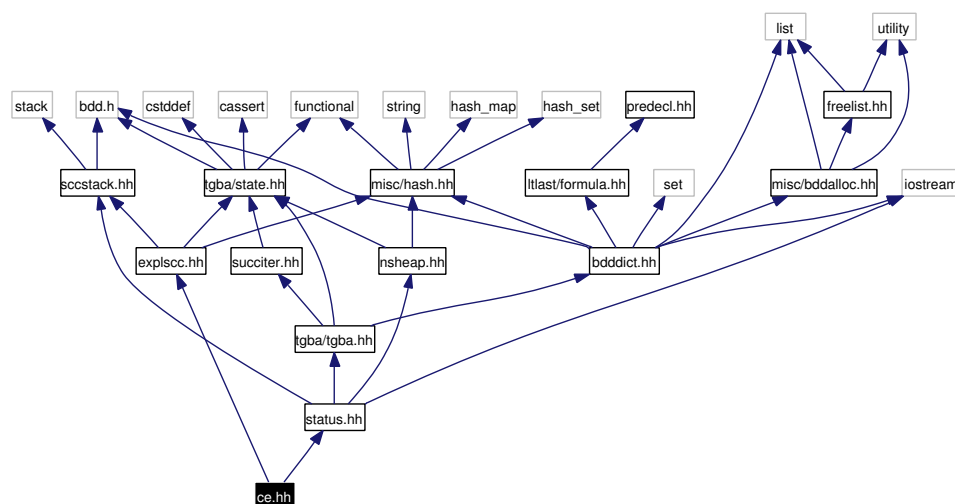
- namespace [spot](#)

## 8.64 tgbalgorithms/gtec/ce.hh File Reference

```
#include "status.hh"
```

```
#include "explscc.hh"
```

Include dependency graph for ce.hh:



This graph shows which files directly or indirectly include this file:



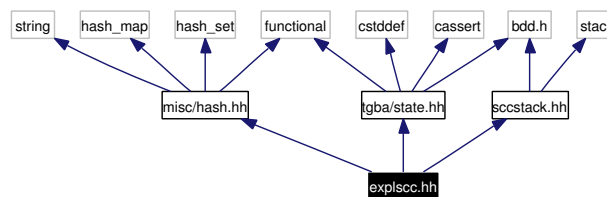
## Namespaces

- namespace [spot](#)

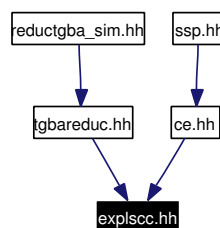
## 8.65 tgbaalgos/gtec/explsc.h File Reference

```
#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:



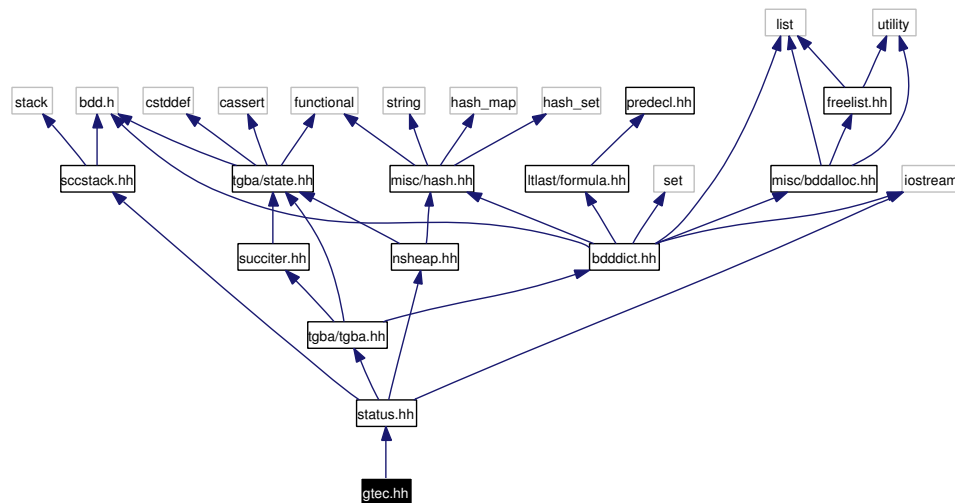
## Namespaces

- namespace [spot](#)

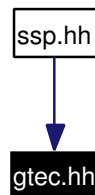
## 8.66 tgbaalgos/gtec/gtec.h File Reference

```
#include "status.h"
```

Include dependency graph for gtec.hh:



This graph shows which files directly or indirectly include this file:



## Namespaces

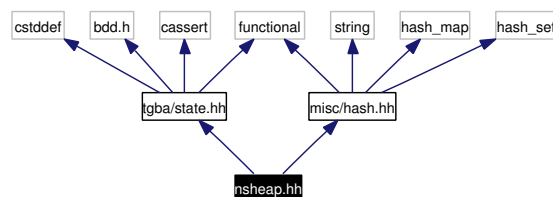
- namespace [spot](#)

## 8.67 tgbaalgos/gtec/nsheap.hh File Reference

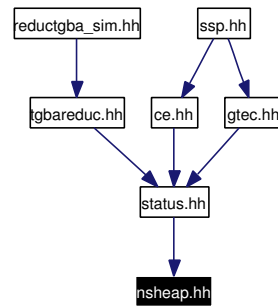
```
#include "tgba/state.hh"
```

```
#include "misc/hash.hh"
```

Include dependency graph for nsheap.hh:



This graph shows which files directly or indirectly include this file:



### Namespaces

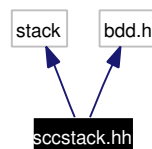
- namespace `spot`

## 8.68 tgbaalgos/gtec/sccstack.hh File Reference

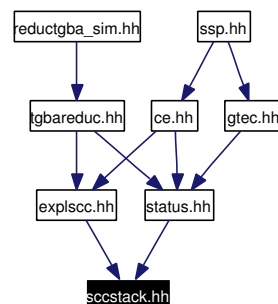
```
#include <stack>
```

```
#include <bdd.h>
```

Include dependency graph for `sccstack.hh`:



This graph shows which files directly or indirectly include this file:



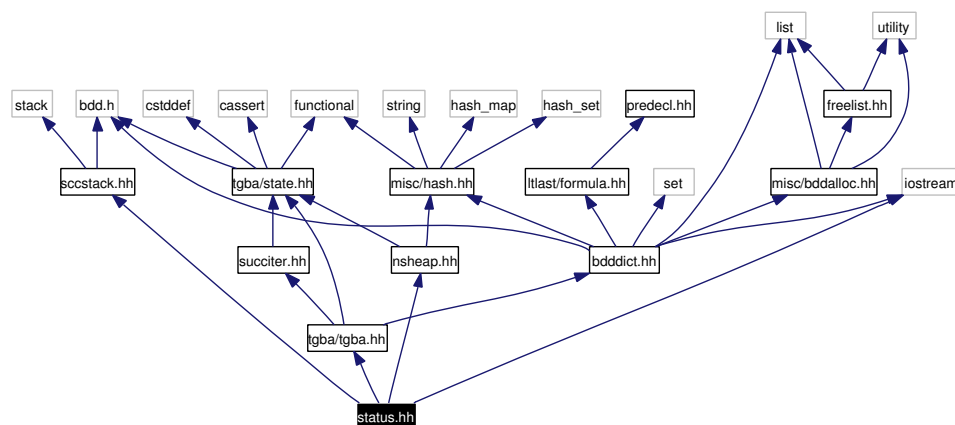
### Namespaces

- namespace `spot`

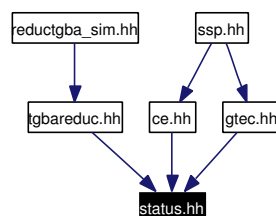
## 8.69 tgbaalgos/gtec/status.hh File Reference

```
#include "sccstack.hh"
#include "nsheap.hh"
#include "tgba/tgba.hh"
#include <iostream>
```

Include dependency graph for status.hh:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [spot](#)

## 8.70 tgbaalgos/lbtt.hh File Reference

```
#include "tgba/tgba.hh"
#include <iostream>
```

Include dependency graph for lbtt.hh:



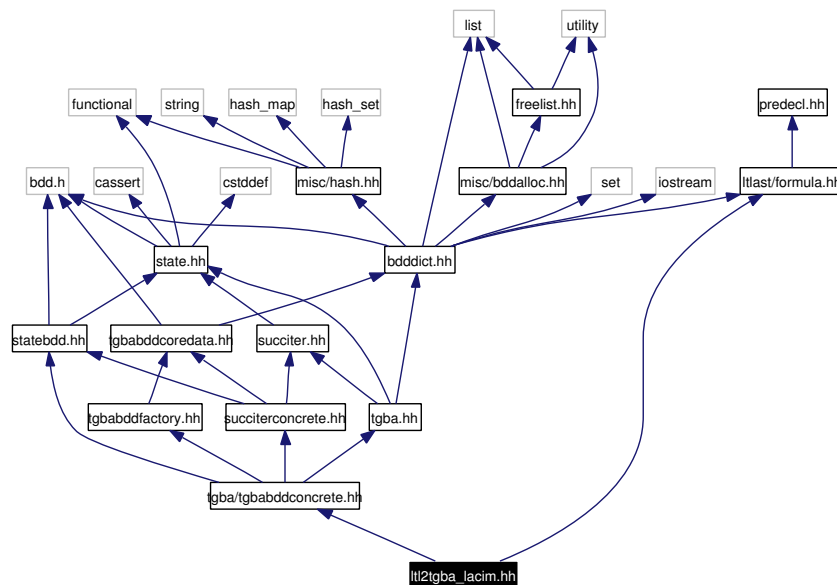


## 8.72 tgbaalgos/ltl2tgba\_lacim.hh File Reference

```
#include "ltlast/formula.hh"
```

```
#include "tgba/tgbabddconcrete.hh"
```

Include dependency graph for ltl2tgba\_lacim.hh:



### Namespaces

- namespace [spot](#)

## 8.73 tgbaalgos/magic.hh File Reference

```
#include "misc/hash.hh"
```

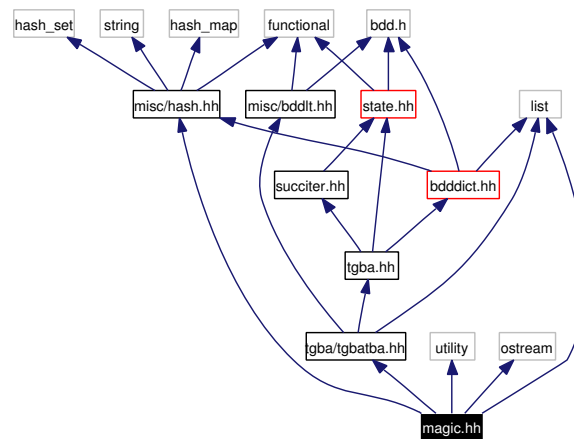
```
#include <list>
```

```
#include <utility>
```

```
#include <ostream>
```

```
#include "tgba/tgbatba.hh"
```

Include dependency graph for magic.hh:



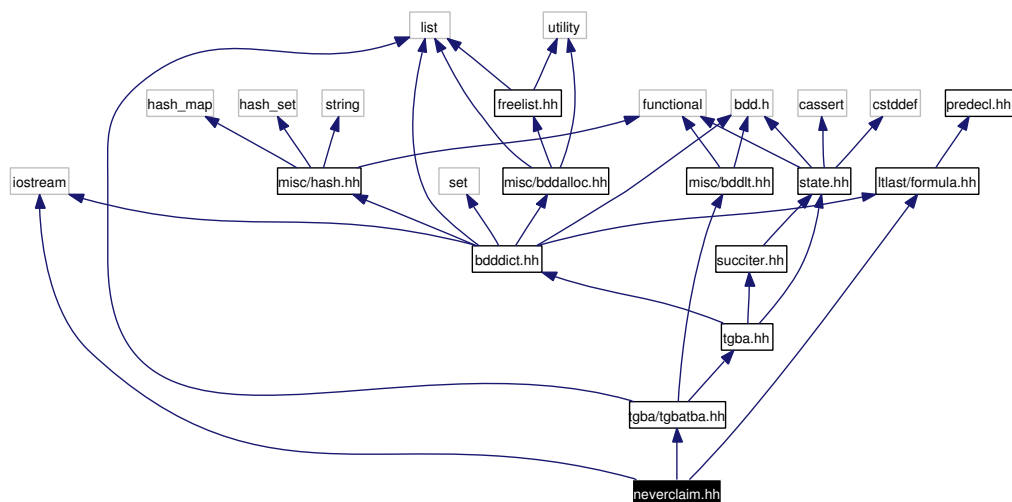
## Namespaces

- namespace [spot](#)

## 8.74 tgbalgorithms/neverclaim.hh File Reference

```
#include <iostream>
#include "ltlast/formula.hh"
#include "tgba/tgbatba.hh"
```

Include dependency graph for neverclaim.hh:



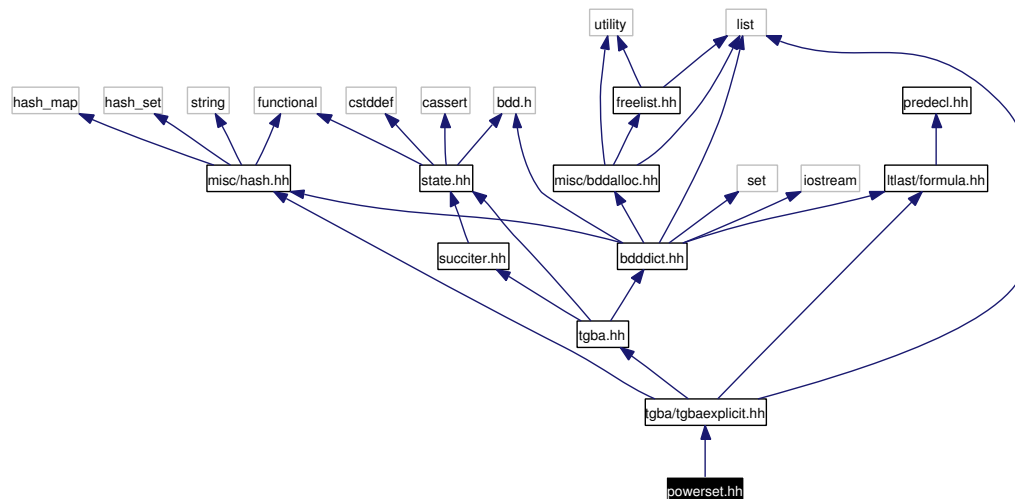
## Namespaces

- namespace [spot](#)

### 8.75 tgbaalgorithms/powerset.hh File Reference

```
#include "tgba/tgbaexplicit.hh"
```

Include dependency graph for powerset.hh:



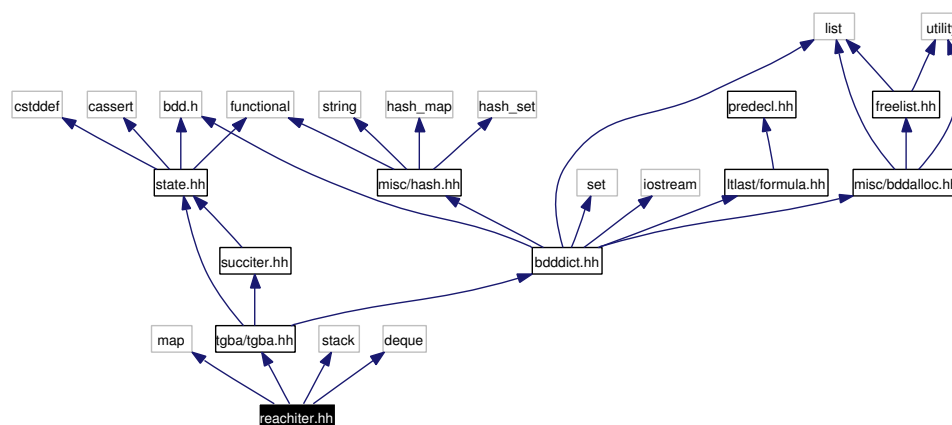
## Namespaces

- namespace **spot**

### 8.76 tgbaalgorithms/reachiter.hh File Reference

```
#include <map>
#include "tgba/tgba.hh"
#include <stack>
#include <deque>
```

Include dependency graph for reachiter.hh:



```

graph TD
    A[reductgba_sim.hh] --> C[reachiter.hh]
    B[tgbareduc.hh] --> C

```

- namespace **spot**

```
#include "tgba/tgbareduc.hh"
#include "tgbaalgos/reachiter.hh"
#include <vector>
#include <list>
#include <sstream>
```

The diagram illustrates a network of dependencies between C++ header files. At the base is 'reductgba\_sim.hh', which depends on 'vector', 'sstream', and 'gbalgbareduc.hh'. Above it, 'gbalgbareduc.hh' has dependencies on 'tgbaexplicit.hh', 'tgbaalgs/gtec/status.hh', and 'gbalgbareduc.hh'. Further up, 'tgbaexplicit.hh' depends on 'list', 'ltlast/formula.hh', 'iostream', 'misc/hash.hh', 'nsheap.hh', 'succiter.hh', 'tgbaalgs/gtec/explicsc.hh', and 'deque'. Other nodes like 'list' depend on 'string', 'hash\_map', 'hash\_set', 'functional', 'bdd.h', and 'stack'. The 'bdddict.hh' node is specifically highlighted with a red border.

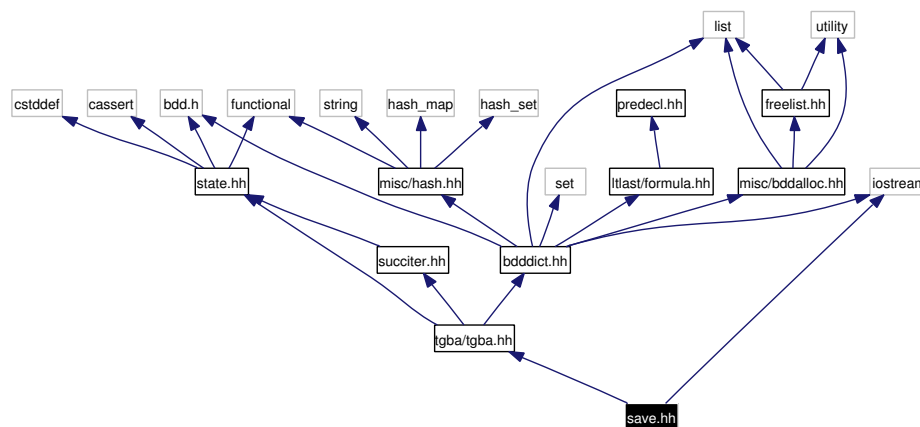
- namespace **spot**

## 8.78 tgbalgorithms/save.hh File Reference

```
#include "tgba/tgba.hh"
```

```
#include <iostream>
```

Include dependency graph for save.hh:



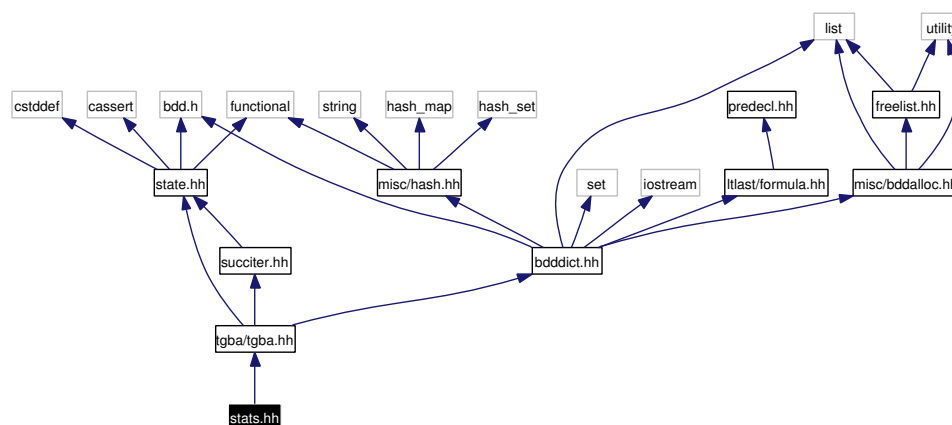
### Namespaces

- namespace [spot](#)

## 8.79 tgbalgorithms/stats.hh File Reference

```
#include "tgba/tgba.hh"
```

Include dependency graph for stats.hh:



### Namespaces

- namespace [spot](#)

## Index

/home/adl/proj/spot/doc/mainpage.dox, [283](#)

~atomic\_prop

spot::ltl::atomic\_prop, [38](#)

~bdd\_dict

spot::bdd\_dict, [49](#)

~binop

spot::ltl::binop, [60](#)

~clone\_visitor

spot::ltl::clone\_visitor, [64](#)

~connected\_component\_hash\_set

spot::connected\_component\_hash\_set, [67](#)

~connected\_component\_hash\_set\_factory

spot::connected\_component\_hash\_set\_  
factory, [69](#)

~constant

spot::ltl::constant, [72](#)

~declarative\_environment

spot::ltl::declarative\_environment, [78](#)

~default\_environment

spot::ltl::default\_environment, [80](#)

~duplicator\_node

spot::duplicator\_node, [83](#)

~duplicator\_node\_delayed

spot::duplicator\_node\_delayed, [87](#)

~emptiness\_check

spot::emptiness\_check, [92](#)

~emptiness\_check\_shy

spot::emptiness\_check\_shy, [94](#)

~emptiness\_check\_status

spot::emptiness\_check\_status, [98](#)

~environment

spot::ltl::environment, [99](#)

~explicit\_connected\_component

spot::explicit\_connected\_component, [101](#)

~explicit\_connected\_component\_factory

spot::explicit\_connected\_component\_  
factory, [102](#)

~formula

spot::ltl::formula, [104](#)

~free\_list

spot::free\_list, [107](#)

~gspn\_interface

spot::gspn\_interface, [110](#)

~gspn\_ssp\_interface

spot::gspn\_ssp\_interface, [111](#)

~magic\_search

spot::magic\_search, [116](#)

~multop

spot::ltl::multop, [125](#)

~numbered\_state\_heap

spot::numbered\_state\_heap, [129](#)

~numbered\_state\_heap\_const\_iterator

spot::numbered\_state\_heap\_const\_iterator,  
[131](#)

~numbered\_state\_heap\_factory

spot::numbered\_state\_heap\_factory, [132](#)

~numbered\_state\_heap\_hash\_map

spot::numbered\_state\_heap\_hash\_map, [134](#)

~numbered\_state\_heap\_hash\_map\_factory

spot::numbered\_state\_heap\_hash\_map\_  
factory, [137](#)

~parity\_game\_graph

spot::parity\_game\_graph, [141](#)

~parity\_game\_graph\_delayed

spot::parity\_game\_graph\_delayed, [146](#)

~parity\_game\_graph\_direct

spot::parity\_game\_graph\_direct, [152](#)

~postfix\_visitor

spot::ltl::postfix\_visitor, [158](#)

~ref\_formula

spot::ltl::ref\_formula, [161](#)

~simplify\_f\_g\_visitor

spot::ltl::simplify\_f\_g\_visitor, [167](#)

~spoiler\_node

spot::spoiler\_node, [170](#)

~spoiler\_node\_delayed

spot::spoiler\_node\_delayed, [173](#)

~state

spot::state, [178](#)

~state\_explicit

spot::state\_explicit, [182](#)

~state\_product

spot::state\_product, [185](#)

~tgba

spot::tgba, [191](#)

~tgba\_bdd\_concrete

spot::tgba\_bdd\_concrete, [197](#)

~tgba\_bdd\_concrete\_factory

spot::tgba\_bdd\_concrete\_factory, [202](#)

~tgba\_explicit

spot::tgba\_explicit, [213](#)

~tgba\_explicit\_succ\_iterator

spot::tgba\_explicit\_succ\_iterator, [220](#)

~tgba\_product

spot::tgba\_product, [225](#)

~tgba\_reachable\_iterator

spot::tgba\_reachable\_iterator, [230](#)

~tgba\_reduc

spot::tgba\_reduc, [246](#)

~tgba\_succ\_iterator

- spot::tgba\_succ\_iterator, 255
- ~tgba\_succ\_iterator\_concrete
  - spot::tgba\_succ\_iterator\_concrete, 259
- ~tgba\_succ\_iterator\_product
  - spot::tgba\_succ\_iterator\_product, 263
- ~tgba\_tba\_proxy
  - spot::tgba\_tba\_proxy, 268
- ~unabbreviate\_logic\_visitor
  - spot::ltl::unabbreviate\_logic\_visitor, 273
- ~unabbreviate\_ltl\_visitor
  - spot::ltl::unabbreviate\_ltl\_visitor, 276
- ~unop
  - spot::ltl::unop, 280
- a
  - spot::magic\_search, 116
- a\_
  - spot::tgba\_tba\_proxy, 270
- acc
  - spot::emptiness\_check\_shy::successor, 96
- acc\_
  - spot::duplicator\_node, 84
  - spot::duplicator\_node\_delayed, 89
  - spot::tgba\_bdd\_concrete\_factory, 203
  - spot::tgba\_reduc, 252
- acc\_cycle\_
  - spot::tgba\_tba\_proxy, 270
- acc\_formula\_map
  - spot::bdd\_dict, 52
- acc\_map
  - spot::bdd\_dict, 52
- acc\_map\_
  - spot::tgba\_bdd\_concrete\_factory, 202
- acc\_set
  - spot::tgba\_bdd\_core\_data, 206
- accept
  - spot::ltl::atomic\_prop, 38
  - spot::ltl::binop, 60
  - spot::ltl::constant, 72
  - spot::ltl::formula, 104
  - spot::ltl::multop, 125
  - spot::ltl::ref\_formula, 161
  - spot::ltl::unop, 280
- acceptance\_condition\_visited\_
  - spot::spoiler\_node\_delayed, 175
- acceptance\_conditions
  - spot::tgba\_bdd\_core\_data, 206
  - spot::tgba\_explicit::transition, 218
- accepting\_path
  - spot::counter\_example, 75
- add\_acceptance\_condition
  - spot::tgba\_explicit, 214
  - spot::tgba\_reduc, 246
- add\_acceptance\_conditions
  - spot::tgba\_explicit, 214
  - spot::tgba\_reduc, 246
- add\_condition
  - spot::tgba\_explicit, 214
  - spot::tgba\_reduc, 246
- add\_conditions
  - spot::tgba\_explicit, 214
  - spot::tgba\_reduc, 246
- add\_duplicator\_node\_delayed
  - spot::parity\_game\_graph\_delayed, 146
- add\_pred
  - spot::duplicator\_node, 83
  - spot::duplicator\_node\_delayed, 87
  - spot::spoiler\_node, 170
  - spot::spoiler\_node\_delayed, 174
- add\_spoiler\_node\_delayed
  - spot::parity\_game\_graph\_delayed, 146
- add\_state
  - spot::parity\_game\_graph, 141
  - spot::parity\_game\_graph\_delayed, 146
  - spot::parity\_game\_graph\_direct, 153
  - spot::tgba\_explicit, 214
  - spot::tgba\_reachable\_iterator, 230
  - spot::tgba\_reachable\_iterator\_breadth\_first, 234
  - spot::tgba\_reachable\_iterator\_depth\_first, 238
  - spot::tgba\_reduc, 246
- add\_succ
  - spot::duplicator\_node, 83
  - spot::duplicator\_node\_delayed, 87
  - spot::spoiler\_node, 170
  - spot::spoiler\_node\_delayed, 174
- all\_acceptance\_conditions
  - spot::tgba, 191
  - spot::tgba\_bdd\_concrete, 197
  - spot::tgba\_bdd\_core\_data, 207
  - spot::tgba\_explicit, 214
  - spot::tgba\_product, 225
  - spot::tgba\_reduc, 247
  - spot::tgba\_tba\_proxy, 268
- all\_acceptance\_conditions\_
  - spot::tgba\_explicit, 217
  - spot::tgba\_explicit\_succ\_iterator, 221
  - spot::tgba\_product, 227
  - spot::tgba\_reduc, 252
- all\_acceptance\_conditions\_computed\_
  - spot::tgba\_explicit, 217
  - spot::tgba\_reduc, 253
- allocate\_variables
  - spot::bdd\_allocator, 43
  - spot::bdd\_dict, 50
- And
  - spot::ltl::multop, 125

- annon\_free\_list
  - spot::bdd\_dict::annon\_free\_list, 55
- arc
  - spot::emptiness\_check\_shy, 95
- as\_bdd
  - spot::state\_bdd, 180
- assert\_emptiness
  - spot::bdd\_dict, 50
- atomic\_prop
  - spot::ltl::atomic\_prop, 38
- aut
  - spot::emptiness\_check\_status, 98
- automata\_
  - spot::parity\_game\_graph, 142
  - spot::parity\_game\_graph\_delayed, 148
  - spot::parity\_game\_graph\_direct, 154
  - spot::tgba\_reachable\_iterator, 231
  - spot::tgba\_reachable\_iterator\_breadth\_first, 235
  - spot::tgba\_reachable\_iterator\_depth\_first, 239
  - spot::tgba\_reduc, 253
- automaton
  - spot::gspn\_interface, 110
  - spot::gspn\_ssp\_interface, 112
- basic\_reduce
  - spot::ltl, 28
- bdd\_allocator
  - spot::bdd\_allocator, 43
- bdd\_dict
  - spot::bdd\_dict, 49
- bdd\_format\_accset
  - spot, 18
- bdd\_format\_formula
  - spot, 18
- bdd\_format\_sat
  - spot, 18
- bdd\_format\_set
  - spot, 18
- bdd\_print\_acc
  - spot, 18
- bdd\_print\_accset
  - spot, 19
- bdd\_print\_dot
  - spot, 19
- bdd\_print\_formula
  - spot, 19
- bdd\_print\_sat
  - spot, 19
- bdd\_print\_set
  - spot, 20
- bdd\_print\_table
  - spot, 20
- bdd\_to\_formula
  - spot, 20
- bdd\_v
  - spot::parity\_game\_graph\_delayed, 146
- begin
  - yy::Location, 113
  - yy::Stack, 176
- binop
  - spot::ltl::binop, 60
- build
  - spot::connected\_component\_hash\_set\_factory, 69
  - spot::explicit\_connected\_component\_factory, 102
  - spot::numbered\_state\_heap\_factory, 132
  - spot::numbered\_state\_heap\_hash\_map\_factory, 137
- build\_graph
  - spot::parity\_game\_graph, 141
  - spot::parity\_game\_graph\_delayed, 147
  - spot::parity\_game\_graph\_direct, 153
- build\_link
  - spot::parity\_game\_graph\_direct, 153
- build\_recurse\_successor\_duplicator
  - spot::parity\_game\_graph\_delayed, 147
- build\_recurse\_successor\_spoiler
  - spot::parity\_game\_graph\_delayed, 147
- build\_sub\_set\_acc\_cond
  - spot::parity\_game\_graph\_delayed, 147
- check
  - spot::emptiness\_check, 92
  - spot::emptiness\_check\_shy, 95
  - spot::magic\_search, 116
- child
  - spot::ltl::unop, 280
- child\_
  - spot::ltl::unop, 281
- children\_
  - spot::ltl::multop, 127
- clone
  - spot::ltl, 28
  - spot::state, 178
  - spot::state\_bdd, 180
  - spot::state\_explicit, 183
  - spot::state\_product, 186
- clone\_visitor
  - spot::ltl::clone\_visitor, 64
- column
  - yy::Position, 156
- columns
  - yy::Location, 113
  - yy::Position, 156
- compare



- spot::duplicator\_node, 83
- spot::duplicator\_node\_delayed, 87
- spot::spoiler\_node, 170
- spot::spoiler\_node\_delayed, 174
- spot::state, 178
- spot::state\_bdd, 180
- spot::state\_explicit, 183
- spot::state\_product, 186
- complement\_all\_acceptance\_conditions
  - spot::tgba\_explicit, 214
  - spot::tgba\_reduc, 247
- complete\_cycle
  - spot::counter\_example, 75
- compute\_scc
  - spot::tgba\_reduc, 247
- compute\_support\_conditions
  - spot::tgba, 191
  - spot::tgba\_bdd\_concrete, 197
  - spot::tgba\_explicit, 214
  - spot::tgba\_product, 225
  - spot::tgba\_reduc, 247
  - spot::tgba\_tba\_proxy, 268
- compute\_support\_variables
  - spot::tgba, 192
  - spot::tgba\_bdd\_concrete, 197
  - spot::tgba\_explicit, 214
  - spot::tgba\_product, 225
  - spot::tgba\_reduc, 247
  - spot::tgba\_tba\_proxy, 268
- condition
  - spot::connected\_component\_hash\_set, 67
  - spot::explicit\_connected\_component, 101
  - spot::scc\_stack::connected\_component, 164
  - spot::tgba\_explicit::transition, 218
- connected\_component
  - spot::scc\_stack::connected\_component, 164
- connected\_component\_hash\_set\_factory
  - spot::connected\_component\_hash\_set\_factory, 69
- constant
  - spot::ltl::constant, 72
- ConstIterator
  - yy::Stack, 176
- constrain\_relation
  - spot::tgba\_bdd\_concrete\_factory, 202
- counter\_example
  - spot::counter\_example, 75
- counter\_example\_ssp
  - spot, 20
- create\_atomic\_prop
  - spot::tgba\_bdd\_concrete\_factory, 202
- create\_state
  - spot::tgba\_bdd\_concrete\_factory, 202
- create\_transition
  - spot::tgba\_explicit, 214
  - spot::tgba\_reduc, 247
- cube\_
  - spot::minato\_isop, 120
- current\_
  - spot::tgba\_succ\_iterator\_concrete, 260
- current\_acc\_
  - spot::tgba\_succ\_iterator\_concrete, 260
- current\_acceptance\_conditions
  - spot::tgba\_explicit\_succ\_iterator, 220
  - spot::tgba\_succ\_iterator, 255
  - spot::tgba\_succ\_iterator\_concrete, 259
  - spot::tgba\_succ\_iterator\_product, 263
- current\_cond\_
  - spot::tgba\_succ\_iterator\_product, 264
- current\_condition
  - spot::tgba\_explicit\_succ\_iterator, 220
  - spot::tgba\_succ\_iterator, 255
  - spot::tgba\_succ\_iterator\_concrete, 259
  - spot::tgba\_succ\_iterator\_product, 263
- current\_state
  - spot::tgba\_explicit\_succ\_iterator, 220
  - spot::tgba\_succ\_iterator, 256
  - spot::tgba\_succ\_iterator\_concrete, 260
  - spot::tgba\_succ\_iterator\_product, 263
- current\_state\_
  - spot::tgba\_succ\_iterator\_concrete, 261
- cycle\_list
  - spot::tgba\_tba\_proxy, 268
- cycle\_path
  - spot::counter\_example, 75
- data\_
  - spot::tgba\_bdd\_concrete, 200
  - spot::tgba\_bdd\_concrete\_factory, 203
  - spot::tgba\_succ\_iterator\_concrete, 261
- dead\_
  - spot::gspn\_interface, 110
- declarative\_environment
  - spot::ltl::declarative\_environment, 78
- declare
  - spot::ltl::declarative\_environment, 78
- declare\_acceptance\_condition
  - spot::tgba\_bdd\_concrete\_factory, 203
  - spot::tgba\_bdd\_core\_data, 206
  - spot::tgba\_explicit, 215
  - spot::tgba\_reduc, 247
- declare\_atomic\_prop
  - spot::tgba\_bdd\_core\_data, 206
- declare\_now\_next
  - spot::tgba\_bdd\_core\_data, 206
- default\_environment

- spot::ltl::default\_environment, 80
- del\_pred
  - spot::duplicator\_node, 83
  - spot::duplicator\_node\_delayed, 88
  - spot::spoiler\_node, 170
  - spot::spoiler\_node\_delayed, 174
- del\_succ
  - spot::duplicator\_node, 83
  - spot::duplicator\_node\_delayed, 88
  - spot::spoiler\_node, 170
  - spot::spoiler\_node\_delayed, 174
- delete\_scc
  - spot::tgba\_reduc, 247
- dest
  - spot::tgba\_explicit::transition, 218
- destroy
  - spot::ltl, 28
- dict
  - spot::tgba\_bdd\_core\_data, 207
- dict\_
  - spot::bdd\_dict::annon\_free\_list, 56
  - spot::gspn\_interface, 110
  - spot::gspn\_ssp\_interface, 112
  - spot::tgba\_explicit, 217
  - spot::tgba\_product, 227
  - spot::tgba\_reduc, 253
- display\_rel\_sim
  - spot::tgba\_reduc, 248
- display\_scc
  - spot::tgba\_reduc, 248
- dn\_v
  - spot, 17
- doit
  - spot::ltl::postfix\_visitor, 158
- doit\_default
  - spot::ltl::postfix\_visitor, 158
- done
  - spot::numbered\_state\_heap\_const\_iterator, 131
  - spot::tgba\_explicit\_succ\_iterator, 220
  - spot::tgba\_succ\_iterator, 256
  - spot::tgba\_succ\_iterator\_concrete, 260
  - spot::tgba\_succ\_iterator\_product, 263
- dotty
  - spot::ltl, 28
- dotty\_reachable
  - spot, 20
- dump
  - spot::bdd\_dict, 50
  - spot::ltl, 28
- dump\_free\_list
  - spot::bdd\_allocator, 43
  - spot::bdd\_dict, 50
  - spot::bdd\_dict::annon\_free\_list, 55
  - spot::free\_list, 107
- dump\_instances
  - spot::ltl::atomic\_prop, 38
- duplicator\_node
  - spot::duplicator\_node, 83
- duplicator\_node\_delayed
  - spot::duplicator\_node\_delayed, 87
- duplicator\_vertice\_
  - spot::parity\_game\_graph, 142
  - spot::parity\_game\_graph\_delayed, 148
  - spot::parity\_game\_graph\_direct, 154
- ecs\_
  - spot::counter\_example, 76
  - spot::emptiness\_check, 92
  - spot::emptiness\_check\_shy, 95
- emptiness\_check
  - spot::emptiness\_check, 92
- emptiness\_check\_shy
  - spot::emptiness\_check\_shy, 94
- emptiness\_check\_ssp\_semi
  - spot, 20
- emptiness\_check\_ssp\_shy
  - spot, 20
- emptiness\_check\_ssp\_shy\_semi
  - spot, 20
- emptiness\_check\_status
  - spot::emptiness\_check\_status, 98
- empty
  - spot::scc\_stack, 163
- end
  - spot::parity\_game\_graph, 141
  - spot::parity\_game\_graph\_delayed, 147
  - spot::parity\_game\_graph\_direct, 153
  - spot::tgba\_reachable\_iterator, 230
  - spot::tgba\_reachable\_iterator\_breadth\_first, 234
  - spot::tgba\_reachable\_iterator\_depth\_first, 238
  - spot::tgba\_reduc, 248
  - yy::Location, 113
  - yy::Stack, 176
- env
  - spot::ltl::atomic\_prop, 38
- env\_
  - spot::gspn\_interface, 110
  - spot::gspn\_ssp\_interface, 112
  - spot::ltl::atomic\_prop, 39
- Equiv
  - spot::ltl::binop, 60
- err\_
  - spot::gspn\_exeption, 108
- escape\_str
  - spot, 20

- extend
  - spot::bdd\_allocator, 43
  - spot::bdd\_dict::annon\_free\_list, 55
  - spot::free\_list, 107
- extvarnum
  - spot::bdd\_allocator, 43
- F
  - spot::ltl::unop, 280
- f0\_max
  - spot::minato\_isop::local\_vars, 121
- f0\_min
  - spot::minato\_isop::local\_vars, 121
- f1\_max
  - spot::minato\_isop::local\_vars, 121
- f1\_min
  - spot::minato\_isop::local\_vars, 121
- f\_max
  - spot::minato\_isop::local\_vars, 121
- f\_min
  - spot::minato\_isop::local\_vars, 121
- False
  - spot::ltl::constant, 72
- false\_instance
  - spot::ltl::constant, 72
- filename
  - yy::Position, 156
- find
  - spot::numbered\_state\_heap, 129
  - spot::numbered\_state\_heap\_hash\_map, 134
- find\_state
  - spot::emptiness\_check\_shy, 95
- finish
  - spot::tgba\_bdd\_concrete\_factory, 203
- first
  - spot::ltl::binop, 61
  - spot::numbered\_state\_heap\_const\_iterator, 131
  - spot::tgba\_explicit\_succ\_iterator, 221
  - spot::tgba\_succ\_iterator, 256
  - spot::tgba\_succ\_iterator\_concrete, 260
  - spot::tgba\_succ\_iterator\_product, 264
- first\_
  - spot::ltl::binop, 62
- FirstStep
  - spot::minato\_isop::local\_vars, 120
- fl
  - spot::bdd\_allocator, 44
  - spot::bdd\_dict::annon\_free\_list, 56
  - spot::free\_list, 108
- format\_parse\_errors
  - spot::ltl, 29
- format\_state
  - spot::tgba, 192
- spot::tgba\_bdd\_concrete, 197
- spot::tgba\_explicit, 215
- spot::tgba\_product, 225
- spot::tgba\_reduc, 248
- spot::tgba\_tba\_proxy, 268
- format\_tgba\_parse\_errors
  - spot, 21
- formula\_to\_bdd
  - spot, 21
- FourthStep
  - spot::minato\_isop::local\_vars, 121
- free\_anonymous\_list\_of
  - spot::bdd\_dict, 52
- free\_anonymous\_list\_of\_type
  - spot::bdd\_dict, 49
- free\_list\_type
  - spot::bdd\_allocator, 42
  - spot::bdd\_dict::annon\_free\_list, 55
  - spot::free\_list, 106
- free\_relation\_simulation
  - spot, 21
- fv\_map
  - spot::bdd\_dict, 49
- G
  - spot::ltl::unop, 280
- g0
  - spot::minato\_isop::local\_vars, 121
- g1
  - spot::minato\_isop::local\_vars, 121
- get\_acc
  - spot::duplicator\_node, 83
  - spot::duplicator\_node\_delayed, 88
- get\_acceptance\_condition
  - spot::tgba\_explicit, 215
  - spot::tgba\_reduc, 248
- get\_acceptance\_condition\_visited
  - spot::spoiler\_node\_delayed, 174
- get\_core\_data
  - spot::tgba\_bdd\_concrete, 197
  - spot::tgba\_bdd\_concrete\_factory, 203
  - spot::tgba\_bdd\_factory, 209
- get\_delayed\_relation\_simulation
  - spot, 21
- get\_dict
  - spot::tgba, 192
  - spot::tgba\_bdd\_concrete, 198
  - spot::tgba\_bdd\_concrete\_factory, 203
  - spot::tgba\_explicit, 215
  - spot::tgba\_product, 225
  - spot::tgba\_reduc, 248
  - spot::tgba\_tba\_proxy, 268
- get\_direct\_relation\_simulation
  - spot, 21

- get\_duplicator\_node
  - spot::duplicator\_node, 83
  - spot::duplicator\_node\_delayed, 88
  - spot::spoiler\_node, 170
  - spot::spoiler\_node\_delayed, 174
- get\_err
  - spot::gspn\_exeption, 108
- get\_index
  - spot::numbered\_state\_heap\_const\_iterator, 131
- get\_init\_bdd
  - spot::tgba\_bdd\_concrete, 198
- get\_init\_state
  - spot::tgba, 192
  - spot::tgba\_bdd\_concrete, 198
  - spot::tgba\_explicit, 215
  - spot::tgba\_product, 225
  - spot::tgba\_reduc, 248
  - spot::tgba\_tba\_proxy, 269
- get\_label
  - spot::duplicator\_node, 83
  - spot::duplicator\_node\_delayed, 88
- get\_lead\_2\_acc\_all
  - spot::duplicator\_node\_delayed, 88
  - spot::spoiler\_node\_delayed, 174
- get\_nb\_state
  - spot::tgba\_reduc, 248
- get\_nb\_succ
  - spot::duplicator\_node, 83
  - spot::duplicator\_node\_delayed, 88
  - spot::spoiler\_node, 171
  - spot::spoiler\_node\_delayed, 174
- get\_nb\_transition
  - spot::tgba\_reduc, 248
- get\_pair
  - spot::duplicator\_node, 84
  - spot::duplicator\_node\_delayed, 88
  - spot::spoiler\_node, 171
  - spot::spoiler\_node\_delayed, 174
- get\_progress\_measure
  - spot::duplicator\_node\_delayed, 88
  - spot::spoiler\_node\_delayed, 174
- get\_prop\_map
  - spot::ltl::declarative\_environment, 78
- get\_relation
  - spot::parity\_game\_graph, 141
  - spot::parity\_game\_graph\_delayed, 147
  - spot::parity\_game\_graph\_direct, 153
- get\_spoiler\_node
  - spot::duplicator\_node, 84
  - spot::duplicator\_node\_delayed, 88
  - spot::spoiler\_node, 171
  - spot::spoiler\_node\_delayed, 174
- get\_state
  - spot::numbered\_state\_heap\_const\_iterator, 131
  - spot::state\_explicit, 183
- get\_where
  - spot::gspn\_exeption, 108
- gspn/common.hh, 283
- gspn/gspn.hh, 284
- gspn/ssp.hh, 284
- gspn\_exeption
  - spot::gspn\_exeption, 108
- gspn\_interface
  - spot::gspn\_interface, 110
- gspn\_ssp\_interface
  - spot::gspn\_ssp\_interface, 111
- h
  - spot::emptiness\_check\_status, 98
  - spot::magic\_search, 116
  - spot::numbered\_state\_heap\_hash\_map, 135
- h\_
  - spot::tgba\_reduc, 253
- has
  - spot::magic\_search, 116
- has\_acceptance\_condition
  - spot::tgba\_explicit, 215
  - spot::tgba\_reduc, 249
- has\_state
  - spot::connected\_component\_hash\_set, 67
  - spot::explicit\_connected\_component, 101
- hash
  - spot::state, 178
  - spot::state\_bdd, 180
  - spot::state\_explicit, 183
  - spot::state\_product, 186
- hash\_type
  - spot::magic\_search, 115
  - spot::numbered\_state\_heap\_hash\_map, 134
- height
  - yy::Stack, 176
- i\_
  - spot::tgba\_explicit\_succ\_iterator, 221
- Implies
  - spot::ltl::binop, 60
- implies
  - spot::duplicator\_node, 84
  - spot::duplicator\_node\_delayed, 88
- implies\_acc
  - spot::duplicator\_node\_delayed, 88
- implies\_label
  - spot::duplicator\_node\_delayed, 88
- index
  - spot::connected\_component\_hash\_set, 67
  - spot::explicit\_connected\_component, 101

- spot::numbered\_state\_heap, 130
- spot::numbered\_state\_heap\_hash\_map, 135
- spot::scc\_stack::connected\_component, 165
- init\_
  - spot::tgba\_bdd\_concrete, 200
  - spot::tgba\_explicit, 217
  - spot::tgba\_reduc, 253
- initial\_column
  - yy::Position, 156
- initial\_line
  - yy::Position, 156
- initialize
  - spot::bdd\_allocator, 43
  - spot::bdd\_dict, 50
- initialized
  - spot::bdd\_allocator, 44
  - spot::bdd\_dict, 52
- insert
  - spot::bdd\_allocator, 43
  - spot::bdd\_dict::annon\_free\_list, 56
  - spot::connected\_component\_hash\_set, 67
  - spot::explicit\_connected\_component, 101
  - spot::free\_list, 107
  - spot::numbered\_state\_heap, 130
  - spot::numbered\_state\_heap\_hash\_map, 135
- instance
  - spot::connected\_component\_hash\_set\_factory, 69
  - spot::ltl::atomic\_prop, 38
  - spot::ltl::binop, 61
  - spot::ltl::default\_environment, 80
  - spot::ltl::multop, 126
  - spot::ltl::unop, 280
  - spot::numbered\_state\_heap\_hash\_map\_factory, 137
- instance\_count
  - spot::ltl::atomic\_prop, 38
  - spot::ltl::binop, 61
  - spot::ltl::multop, 126
  - spot::ltl::unop, 280
- instances
  - spot::ltl::atomic\_prop, 39
  - spot::ltl::binop, 62
  - spot::ltl::multop, 127
  - spot::ltl::unop, 281
- is\_alpha\_ball
  - spot::tgba\_reduc, 249
- is\_eventual
  - spot::ltl, 29
- is\_FG
  - spot::ltl, 29
- is\_GF
  - spot::ltl, 29
- is\_include
  - spot, 21
- is\_not\_accepting
  - spot::tgba\_reduc, 249
- is\_registered\_acceptance\_variable
  - spot::bdd\_dict, 50
- is\_registered\_proposition
  - spot::bdd\_dict, 50
- is\_registered\_state
  - spot::bdd\_dict, 50
- is\_terminal
  - spot::tgba\_reduc, 249
- is\_universal
  - spot::ltl, 29
- Iterator
  - yy::Stack, 176
- iterator
  - spot::numbered\_state\_heap, 130
  - spot::numbered\_state\_heap\_hash\_map, 135
- label\_
  - spot::duplicator\_node, 84
  - spot::duplicator\_node\_delayed, 89
- last\_support\_conditions\_input\_
  - spot::tgba, 194
- last\_support\_conditions\_output\_
  - spot::tgba, 194
- last\_support\_variables\_input\_
  - spot::tgba, 194
- last\_support\_variables\_output\_
  - spot::tgba, 194
- lbtt\_reachable
  - spot, 21
- lead\_2\_acc\_all\_
  - spot::duplicator\_node\_delayed, 89
  - spot::spoiler\_node\_delayed, 175
- left
  - spot::state\_product, 186
- left\_
  - spot::state\_product, 186
  - spot::tgba\_product, 227
  - spot::tgba\_succ\_iterator\_product, 264
- left\_acc\_complement\_
  - spot::tgba\_product, 227
- left\_neg\_
  - spot::tgba\_succ\_iterator\_product, 264
- length
  - spot::ltl, 30
- lift
  - spot::parity\_game\_graph, 141
  - spot::parity\_game\_graph\_delayed, 147
  - spot::parity\_game\_graph\_direct, 153
- line
  - yy::Position, 156

- lines
  - yy::Location, 113
  - yy::Position, 156
- lnode\_pred
  - spot::duplicator\_node, 84
  - spot::duplicator\_node\_delayed, 89
  - spot::spoiler\_node, 171
  - spot::spoiler\_node\_delayed, 175
- lnode\_succ
  - spot::duplicator\_node, 84
  - spot::duplicator\_node\_delayed, 89
  - spot::spoiler\_node, 171
  - spot::spoiler\_node\_delayed, 175
- local\_vars
  - spot::minato\_isop::local\_vars, 121
- Location
  - yy::Location, 113
- ltl\_to\_tgba\_fm
  - spot, 21
- ltl\_to\_tgba\_lacim
  - spot, 22
- ltlast/allnodes.hh, 285
- ltlast/atomic\_prop.hh, 285
- ltlast/binop.hh, 286
- ltlast/constant.hh, 287
- ltlast/formula.hh, 288
- ltlast/multop.hh, 288
- ltlast/predecl.hh, 289
- ltlast/refformula.hh, 289
- ltlast/unop.hh, 290
- ltlast/visitor.hh, 291
- ltlenv/declenv.hh, 291
- ltlenv/defaultenv.hh, 292
- ltlenv/environment.hh, 293
- ltlparse/location.hh, 293
- ltlparse/position.hh, 294
- ltlparse/public.hh, 295
- ltlparse/stack.hh, 297
- ltlvisit/basicreduce.hh, 297
- ltlvisit/clone.hh, 298
- ltlvisit/destroy.hh, 299
- ltlvisit/dotty.hh, 299
- ltlvisit/dump.hh, 300
- ltlvisit/length.hh, 301
- ltlvisit/lunabbrev.hh, 301
- ltlvisit/nenoform.hh, 302
- ltlvisit/postfix.hh, 302
- ltlvisit/reduce.hh, 303
- ltlvisit/simpfg.hh, 303
- ltlvisit/syntimpl.hh, 304
- ltlvisit/tostring.hh, 304
- ltlvisit/tunabbrev.hh, 305
- lvarnum
  - spot::bdd\_allocator, 44
  - spot::bdd\_dict, 52
- m
  - spot::magic\_search::magic\_state, 118
- magic\_search
  - spot::magic\_search, 116
- map
  - spot::ltl::atomic\_prop, 37
  - spot::ltl::binop, 60
  - spot::ltl::multop, 125
  - spot::ltl::unop, 280
- match
  - spot::duplicator\_node, 84
  - spot::duplicator\_node\_delayed, 88
- merge\_state
  - spot::tgba\_reduc, 249
- merge\_transitions
  - spot::tgba\_explicit, 215
  - spot::tgba\_reduc, 249
- minato\_isop
  - spot::minato\_isop, 119
- misc/bddalloc.hh, 305
- misc/bddlt.hh, 306
- misc/escape.hh, 307
- misc/freelist.hh, 307
- misc/hash.hh, 308
- misc/minato.hh, 308
- misc/version.hh, 309
- multop
  - spot::ltl::multop, 125
- name
  - spot::ltl::atomic\_prop, 38
  - spot::ltl::declarative\_environment, 78
  - spot::ltl::default\_environment, 80
  - spot::ltl::environment, 99
- name\_
  - spot::ltl::atomic\_prop, 39
- name\_state\_map\_
  - spot::tgba\_explicit, 217
  - spot::tgba\_reduc, 253
- nb\_node\_parity\_game
  - spot::parity\_game\_graph, 142
  - spot::parity\_game\_graph\_delayed, 148
  - spot::parity\_game\_graph\_direct, 154
- nb\_set\_acc\_cond
  - spot::parity\_game\_graph\_delayed, 147
- neg\_acceptance\_conditions
  - spot::tgba, 192
  - spot::tgba\_bdd\_concrete, 198
  - spot::tgba\_explicit, 215
  - spot::tgba\_product, 226
  - spot::tgba\_reduc, 249
  - spot::tgba\_tba\_proxy, 269

- neg\_acceptance\_conditions\_
  - spot::tgba\_explicit, 217
  - spot::tgba\_product, 227
  - spot::tgba\_reduc, 253
- negacc\_set
  - spot::tgba\_bdd\_core\_data, 207
- negative\_normal\_form
  - spot::ltl, 30
- never\_claim\_reachable
  - spot, 23
- next
  - spot::minato\_isop, 119
  - spot::numbered\_state\_heap\_const\_iterator, 131
  - spot::tgba\_explicit\_succ\_iterator, 221
  - spot::tgba\_succ\_iterator, 256
  - spot::tgba\_succ\_iterator\_concrete, 260
  - spot::tgba\_succ\_iterator\_product, 264
- next\_non\_false\_
  - spot::tgba\_succ\_iterator\_product, 264
- next\_set
  - spot::tgba\_bdd\_core\_data, 207
- next\_state
  - spot::parity\_game\_graph, 141
  - spot::parity\_game\_graph\_delayed, 147
  - spot::parity\_game\_graph\_direct, 153
  - spot::tgba\_reachable\_iterator, 230
  - spot::tgba\_reachable\_iterator\_breadth\_first, 234
  - spot::tgba\_reachable\_iterator\_depth\_first, 238
  - spot::tgba\_reduc, 249
- next\_to\_now
  - spot::bdd\_dict, 52
- nonacceptant\_lbtt\_reachable
  - spot, 23
- Not
  - spot::ltl::unop, 280
- not\_win
  - spot::duplicator\_node, 84
  - spot::duplicator\_node\_delayed, 89
  - spot::spoiler\_node, 171
  - spot::spoiler\_node\_delayed, 175
- notacc\_set
  - spot::tgba\_bdd\_core\_data, 207
- notnext\_set
  - spot::tgba\_bdd\_core\_data, 207
- notnow\_set
  - spot::tgba\_bdd\_core\_data, 207
- notvar\_set
  - spot::tgba\_bdd\_core\_data, 208
- now\_formula\_map
  - spot::bdd\_dict, 52
- now\_map
  - spot::bdd\_dict, 52
- now\_set
  - spot::tgba\_bdd\_core\_data, 208
- now\_to\_next
  - spot::bdd\_dict, 52
- nownext\_set
  - spot::tgba\_bdd\_core\_data, 208
- ns\_map
  - spot::tgba\_explicit, 213
  - spot::tgba\_reduc, 246
- nth
  - spot::ltl::multop, 126
- num
  - spot::emptiness\_check\_shy, 95
- num\_
  - spot::duplicator\_node, 84
  - spot::duplicator\_node\_delayed, 89
  - spot::spoiler\_node, 171
  - spot::spoiler\_node\_delayed, 175
- numbered\_state\_heap\_hash\_map\_factory
  - spot::numbered\_state\_heap\_hash\_map\_factory, 137
- op
  - spot::ltl::binop, 61
  - spot::ltl::multop, 126
  - spot::ltl::unop, 280
- op\_
  - spot::ltl::binop, 62
  - spot::ltl::multop, 127
  - spot::ltl::unop, 281
- op\_name
  - spot::ltl::binop, 61
  - spot::ltl::multop, 126
  - spot::ltl::unop, 281
- operator()
  - spot::bdd\_less\_than, 57
  - spot::ltl::multop::paircmp, 128
  - spot::ptr\_hash, 159
  - spot::state\_ptr\_equal, 187
  - spot::state\_ptr\_hash, 188
  - spot::state\_ptr\_less\_than, 188
  - spot::string\_hash, 189
- operator+
  - yy, 33
- operator+=
  - yy, 33
- operator-
  - yy, 33
- operator-=
  - yy, 33
- operator<<
  - spot, 23
  - yy, 33, 34



- operator=
  - spot::bdd\_dict, 50
  - spot::tgba\_bdd\_core\_data, 206
- operator[]
  - yy::Slice, 168
  - yy::Stack, 176, 177
- Or
  - spot::ltl::multop, 125
- pair
  - spot::ltl::atomic\_prop, 37
  - spot::ltl::binop, 60
  - spot::ltl::multop, 125
  - spot::ltl::unop, 280
- pair\_state\_successors
  - spot::emptiness\_check\_shy, 94
- pairf
  - spot::ltl::binop, 60
- parity\_game\_graph
  - spot::parity\_game\_graph, 141
- parity\_game\_graph\_delayed
  - spot::parity\_game\_graph\_delayed, 146
- parity\_game\_graph\_direct
  - spot::parity\_game\_graph\_direct, 152
- parse
  - spot::ltl, 30
- parse\_error
  - spot::ltl, 28
- parse\_error\_list
  - spot::ltl, 28
- period
  - spot::counter\_example, 76
- pop
  - spot::scc\_stack, 163
  - yy::Stack, 177
- pos\_lenght\_pair
  - spot::bdd\_allocator, 42
  - spot::bdd\_dict::annon\_free\_list, 55
  - spot::free\_list, 106
- Position
  - yy::Position, 156
- postfix\_visitor
  - spot::ltl::postfix\_visitor, 158
- print
  - spot::parity\_game\_graph, 141
  - spot::parity\_game\_graph\_delayed, 147
  - spot::parity\_game\_graph\_direct, 153
- print\_result
  - spot::counter\_example, 76
  - spot::magic\_search, 116
- print\_stats
  - spot::counter\_example, 76
  - spot::emptiness\_check\_status, 98
- process\_link
  - spot::parity\_game\_graph, 141
  - spot::parity\_game\_graph\_delayed, 147
  - spot::parity\_game\_graph\_direct, 153
  - spot::tgba\_reachable\_iterator, 231
  - spot::tgba\_reachable\_iterator\_breadth\_first, 235
  - spot::tgba\_reachable\_iterator\_depth\_first, 239
  - spot::tgba\_reduc, 250
- process\_state
  - spot::parity\_game\_graph, 142
  - spot::parity\_game\_graph\_delayed, 148
  - spot::parity\_game\_graph\_direct, 154
  - spot::tgba\_reachable\_iterator, 231
  - spot::tgba\_reachable\_iterator\_breadth\_first, 235
  - spot::tgba\_reachable\_iterator\_depth\_first, 239
  - spot::tgba\_reduc, 250
- product
  - spot, 23
- progress\_measure\_
  - spot::duplicator\_node\_delayed, 89
  - spot::spoiler\_node\_delayed, 175
- project\_state
  - spot::tgba, 192
  - spot::tgba\_bdd\_concrete, 198
  - spot::tgba\_explicit, 216
  - spot::tgba\_product, 226
  - spot::tgba\_reduc, 250
  - spot::tgba\_tba\_proxy, 269
- prop\_map
  - spot::ltl::declarative\_environment, 77
- props\_
  - spot::ltl::declarative\_environment, 78
- prune
  - spot::duplicator\_node, 84
  - spot::duplicator\_node\_delayed, 88
  - spot::spoiler\_node, 171
  - spot::spoiler\_node\_delayed, 174
- prune\_acc
  - spot::tgba\_reduc, 250
- prune\_automata
  - spot::tgba\_reduc, 250
- prune\_scc
  - spot::tgba\_reduc, 250
- push
  - spot::magic\_search, 116
  - spot::scc\_stack, 163
  - yy::Stack, 177
- quotient\_state
  - spot::tgba\_reduc, 251



- R
  - spot::ltl::binop, 60
- range\_
  - yy::Slice, 168
- recurse
  - spot::ltl::clone\_visitor, 64
  - spot::ltl::simplify\_f\_g\_visitor, 167
  - spot::ltl::unabbreviate\_logic\_visitor, 273
  - spot::ltl::unabbreviate\_ltl\_visitor, 276
- redirect\_transition
  - spot::tgba\_reduc, 251
- reduc\_tgba\_sim
  - spot, 23
- reduce
  - spot::ltl, 30
- Reduce\_All
  - spot, 17
  - spot::ltl, 28
- Reduce\_Basics
  - spot::ltl, 28
- Reduce\_Del\_Sim
  - spot, 17
- Reduce\_Dir\_Sim
  - spot, 17
- Reduce\_Eventuality\_And\_Universality
  - spot::ltl, 28
- Reduce\_None
  - spot, 17
  - spot::ltl, 28
- reduce\_options
  - spot::ltl, 28
- Reduce\_Scc
  - spot, 17
- Reduce\_Syntactic\_Implications
  - spot::ltl, 28
- reduce\_tgba\_options
  - spot, 17
- ref
  - spot::ltl::atomic\_prop, 38
  - spot::ltl::binop, 61
  - spot::ltl::constant, 73
  - spot::ltl::formula, 104
  - spot::ltl::multop, 126
  - spot::ltl::ref\_formula, 162
  - spot::ltl::unop, 281
- ref\_
  - spot::ltl::atomic\_prop, 38
  - spot::ltl::binop, 61
  - spot::ltl::constant, 73
  - spot::ltl::formula, 104
  - spot::ltl::multop, 127
  - spot::ltl::ref\_formula, 162
  - spot::ltl::unop, 281
- ref\_count\_
  - spot::ltl::atomic\_prop, 39
  - spot::ltl::binop, 61
  - spot::ltl::multop, 127
  - spot::ltl::ref\_formula, 162
  - spot::ltl::unop, 281
- ref\_counter\_
  - spot::ltl::ref\_formula, 162
- ref\_formula
  - spot::ltl::ref\_formula, 161
- ref\_set
  - spot::bdd\_dict, 49
- register\_acceptance\_variable
  - spot::bdd\_dict, 50
- register\_acceptance\_variables
  - spot::bdd\_dict, 50
- register\_all\_variables\_of
  - spot::bdd\_dict, 51
- register\_anonymous\_variables
  - spot::bdd\_dict, 51
- register\_n
  - spot::bdd\_allocator, 43
  - spot::bdd\_dict::annon\_free\_list, 56
  - spot::free\_list, 107
- register\_proposition
  - spot::bdd\_dict, 51
- register\_propositions
  - spot::bdd\_dict, 51
- register\_state
  - spot::bdd\_dict, 51
- relation
  - spot::tgba\_bdd\_core\_data, 208
- release\_n
  - spot::bdd\_allocator, 43
  - spot::bdd\_dict::annon\_free\_list, 56
  - spot::free\_list, 107
- release\_variables
  - spot::bdd\_allocator, 44
  - spot::bdd\_dict, 51
- remove
  - spot::bdd\_allocator, 44
  - spot::bdd\_dict::annon\_free\_list, 56
  - spot::free\_list, 107
- remove\_acc
  - spot::tgba\_reduc, 251
- remove\_component
  - spot::emptiness\_check, 92
  - spot::emptiness\_check\_shy, 95
  - spot::tgba\_reduc, 251
- remove\_predecessor\_state
  - spot::tgba\_reduc, 251
- remove\_scc
  - spot::tgba\_reduc, 251
- remove\_scc\_depth\_first
  - spot::tgba\_reduc, 251

- remove\_state
  - spot::tgba\_reduc, 251
- require
  - spot::ltl::declarative\_environment, 78
  - spot::ltl::default\_environment, 80
  - spot::ltl::environment, 99
- result
  - spot::emptiness\_check, 92
  - spot::emptiness\_check\_shy, 95
  - spot::ltl::clone\_visitor, 64
  - spot::ltl::simplify\_f\_g\_visitor, 167
  - spot::ltl::unabbreviate\_logic\_visitor, 273
  - spot::ltl::unabbreviate\_ltl\_visitor, 276
- result\_
  - spot::ltl::clone\_visitor, 65
  - spot::ltl::simplify\_f\_g\_visitor, 167
  - spot::ltl::unabbreviate\_logic\_visitor, 274
  - spot::ltl::unabbreviate\_ltl\_visitor, 277
- ret\_
  - spot::minato\_isop, 120
- right
  - spot::state\_product, 186
- right\_
  - spot::state\_product, 186
  - spot::tgba\_product, 227
  - spot::tgba\_succ\_iterator\_product, 264
- right\_acc\_complement\_
  - spot::tgba\_product, 227
- right\_neg\_
  - spot::tgba\_succ\_iterator\_product, 264
- root
  - spot::emptiness\_check\_status, 98
- root\_
  - spot::tgba\_reduc, 253
- run
  - spot::parity\_game\_graph, 142
  - spot::parity\_game\_graph\_delayed, 148
  - spot::parity\_game\_graph\_direct, 154
  - spot::tgba\_reachable\_iterator, 231
  - spot::tgba\_reachable\_iterator\_breadth\_first, 235
  - spot::tgba\_reachable\_iterator\_depth\_first, 239
  - spot::tgba\_reduc, 251
- s
  - spot::emptiness\_check\_shy::successor, 96
  - spot::magic\_search::magic\_state, 118
  - spot::scc\_stack, 164
- s\_
  - spot::tgba\_explicit\_succ\_iterator, 221
- s\_v
  - spot, 17
- sc\_
  - spot::duplicator\_node, 84
  - spot::duplicator\_node\_delayed, 89
  - spot::spoiler\_node, 171
  - spot::spoiler\_node\_delayed, 175
- scc\_computed\_
  - spot::tgba\_reduc, 253
- second
  - spot::ltl::binop, 61
- second\_
  - spot::ltl::binop, 62
- SecondStep
  - spot::minato\_isop::local\_vars, 120
- seen
  - spot::parity\_game\_graph, 142
  - spot::parity\_game\_graph\_delayed, 148
  - spot::parity\_game\_graph\_direct, 154
  - spot::tgba\_reachable\_iterator, 231
  - spot::tgba\_reachable\_iterator\_breadth\_first, 235
  - spot::tgba\_reachable\_iterator\_depth\_first, 239
  - spot::tgba\_reduc, 253
- seen\_
  - spot::tgba\_reduc, 253
- seen\_map
  - spot::parity\_game\_graph, 140
  - spot::parity\_game\_graph\_delayed, 146
  - spot::parity\_game\_graph\_direct, 152
  - spot::tgba\_reachable\_iterator, 230
  - spot::tgba\_reachable\_iterator\_breadth\_first, 234
  - spot::tgba\_reachable\_iterator\_depth\_first, 238
  - spot::tgba\_reduc, 246
- seen\_with
  - spot::magic\_search::magic, 117
- seen\_without
  - spot::magic\_search::magic, 117
- seq\_
  - yy::Stack, 177
- set\_init\_state
  - spot::tgba\_bdd\_concrete, 199
  - spot::tgba\_explicit, 216
  - spot::tgba\_reduc, 251
- set\_lead\_2\_acc\_all
  - spot::duplicator\_node\_delayed, 88
- set\_type
  - spot::connected\_component\_hash\_set, 67
- set\_win
  - spot::duplicator\_node, 84
  - spot::duplicator\_node\_delayed, 88
  - spot::spoiler\_node, 171
  - spot::spoiler\_node\_delayed, 174
- si\_
  - spot::tgba\_reduc, 253

- spot::tgba\_reduc, 253
- simplify\_f\_g
  - spot::ltl, 31
- simplify\_f\_g\_visitor
  - spot::ltl::simplify\_f\_g\_visitor, 167
- simulation\_relation
  - spot, 17
- size
  - spot::ltl::multop, 127
  - spot::numbered\_state\_heap, 130
  - spot::numbered\_state\_heap\_hash\_map, 135
  - spot::scc\_stack, 163
- Slice
  - yy::Slice, 168
- sn\_map
  - spot::tgba\_explicit, 213
  - spot::tgba\_reduc, 246
- sn\_v
  - spot, 17
- sp\_map
  - spot::tgba\_reduc, 246
- spoiler\_node
  - spot::spoiler\_node, 170
- spoiler\_node\_delayed
  - spot::spoiler\_node\_delayed, 173
- spoiler\_vertice\_
  - spot::parity\_game\_graph, 142
  - spot::parity\_game\_graph\_delayed, 148
  - spot::parity\_game\_graph\_direct, 154
- spot, 11
  - bdd\_format\_accset, 18
  - bdd\_format\_formula, 18
  - bdd\_format\_sat, 18
  - bdd\_format\_set, 18
  - bdd\_print\_acc, 18
  - bdd\_print\_accset, 19
  - bdd\_print\_dot, 19
  - bdd\_print\_formula, 19
  - bdd\_print\_sat, 19
  - bdd\_print\_set, 20
  - bdd\_print\_table, 20
  - bdd\_to\_formula, 20
  - counter\_example\_ssp, 20
  - dn\_v, 17
  - dotty\_reachable, 20
  - emptiness\_check\_ssp\_semi, 20
  - emptiness\_check\_ssp\_shy, 20
  - emptiness\_check\_ssp\_shy\_semi, 20
  - escape\_str, 20
  - format\_tgba\_parse\_errors, 21
  - formula\_to\_bdd, 21
  - free\_relation\_simulation, 21
  - get\_delayed\_relation\_simulation, 21
  - get\_direct\_relation\_simulation, 21
  - is\_include, 21
  - lbt\_reachable, 21
  - ltl\_to\_tgba\_fm, 21
  - ltl\_to\_tgba\_lacim, 22
  - never\_claim\_reachable, 23
  - nonacceptant\_lbt\_reachable, 23
  - operator<<, 23
  - product, 23
  - reduc\_tgba\_sim, 23
  - Reduce\_All, 17
  - Reduce\_Del\_Sim, 17
  - Reduce\_Dir\_Sim, 17
  - Reduce\_None, 17
  - Reduce\_Scc, 17
  - reduce\_tgba\_options, 17
  - s\_v, 17
  - simulation\_relation, 17
  - sn\_v, 17
  - state\_couple, 17
  - stats\_reachable, 23
  - tgba\_dupexp\_bfs, 24
  - tgba\_dupexp\_dfs, 24
  - tgba\_parse, 24
  - tgba\_parse\_error, 17
  - tgba\_parse\_error\_list, 17
  - tgba\_powerset, 24
  - tgba\_save\_reachable, 24
  - version, 24
- spot::bdd\_allocator, 39
  - allocate\_variables, 43
  - bdd\_allocator, 43
  - dump\_free\_list, 43
  - extend, 43
  - extvarnum, 43
  - fl, 44
  - free\_list\_type, 42
  - initialize, 43
  - initialized, 44
  - insert, 43
  - lvarnum, 44
  - pos\_lenght\_pair, 42
  - register\_n, 43
  - release\_n, 43
  - release\_variables, 44
  - remove, 44
- spot::bdd\_dict, 44
  - ~bdd\_dict, 49
  - acc\_formula\_map, 52
  - acc\_map, 52
  - allocate\_variables, 50
  - assert\_emptiness, 50
  - bdd\_dict, 49
  - dump, 50
  - dump\_free\_list, 50

- free\_anonymous\_list\_of, 52
- free\_anonymous\_list\_of\_type, 49
- fv\_map, 49
- initialize, 50
- initialized, 52
- is\_registered\_acceptance\_variable, 50
- is\_registered\_proposition, 50
- is\_registered\_state, 50
- lvarnum, 52
- next\_to\_now, 52
- now\_formula\_map, 52
- now\_map, 52
- now\_to\_next, 52
- operator=, 50
- ref\_set, 49
- register\_acceptance\_variable, 50
- register\_acceptance\_variables, 50
- register\_all\_variables\_of, 51
- register\_anonymous\_variables, 51
- register\_proposition, 51
- register\_propositions, 51
- register\_state, 51
- release\_variables, 51
- unregister\_all\_my\_variables, 52
- unregister\_variable, 52
- var\_formula\_map, 53
- var\_map, 53
- var\_refs, 53
- vf\_map, 49
- vr\_map, 49
- spot::bdd\_dict::anon\_free\_list, 53
  - anon\_free\_list, 55
  - dict\_, 56
  - dump\_free\_list, 55
  - extend, 55
  - fl, 56
  - free\_list\_type, 55
  - insert, 56
  - pos\_lenght\_pair, 55
  - register\_n, 56
  - release\_n, 56
  - remove, 56
- spot::bdd\_less\_than, 56
  - operator(), 57
- spot::connected\_component\_hash\_set, 65
  - ~connected\_component\_hash\_set, 67
  - condition, 67
  - has\_state, 67
  - index, 67
  - insert, 67
  - set\_type, 67
  - states, 67
- spot::connected\_component\_hash\_set\_factory, 67
  - ~connected\_component\_hash\_set\_factory, 69
  - build, 69
  - connected\_component\_hash\_set\_factory, 69
  - instance, 69
- spot::counter\_example, 74
  - accepting\_path, 75
  - complete\_cycle, 75
  - counter\_example, 75
  - cycle\_path, 75
  - ecs\_, 76
  - period, 76
  - print\_result, 76
  - print\_stats, 76
  - state\_proposition, 75
  - state\_sequence, 75
  - suffix, 76
- spot::duplicator\_node, 80
  - ~duplicator\_node, 83
  - acc\_, 84
  - add\_pred, 83
  - add\_succ, 83
  - compare, 83
  - del\_pred, 83
  - del\_succ, 83
  - duplicator\_node, 83
  - get\_acc, 83
  - get\_duplicator\_node, 83
  - get\_label, 83
  - get\_nb\_succ, 83
  - get\_pair, 84
  - get\_spoiler\_node, 84
  - implies, 84
  - label\_, 84
  - lnode\_pred, 84
  - lnode\_succ, 84
  - match, 84
  - not\_win, 84
  - num\_, 84
  - prune, 84
  - sc\_, 84
  - set\_win, 84
  - succ\_to\_string, 84
  - to\_string, 84
- spot::duplicator\_node\_delayed, 85
  - ~duplicator\_node\_delayed, 87
  - acc\_, 89
  - add\_pred, 87
  - add\_succ, 87
  - compare, 87
  - del\_pred, 88
  - del\_succ, 88
  - duplicator\_node\_delayed, 87

- get\_acc, 88
- get\_duplicator\_node, 88
- get\_label, 88
- get\_lead\_2\_acc\_all, 88
- get\_nb\_succ, 88
- get\_pair, 88
- get\_progress\_measure, 88
- get\_spoiler\_node, 88
- implies, 88
- implies\_acc, 88
- implies\_label, 88
- label\_, 89
- lead\_2\_acc\_all\_, 89
- lnode\_pred, 89
- lnode\_succ, 89
- match, 88
- not\_win, 89
- num\_, 89
- progress\_measure\_, 89
- prune, 88
- sc\_, 89
- set\_lead\_2\_acc\_all, 88
- set\_win, 88
- succ\_to\_string, 88
- to\_string, 89
- spot::emptiness\_check, 89
  - ~emptiness\_check, 92
  - check, 92
  - ecs\_, 92
  - emptiness\_check, 92
  - remove\_component, 92
  - result, 92
- spot::emptiness\_check\_shy, 92
  - ~emptiness\_check\_shy, 94
  - arc, 95
  - check, 95
  - ecs\_, 95
  - emptiness\_check\_shy, 94
  - find\_state, 95
  - num, 95
  - pair\_state\_successors, 94
  - remove\_component, 95
  - result, 95
  - succ\_queue, 94
  - todo, 95
- spot::emptiness\_check\_shy::successor, 95
  - acc, 96
  - s, 96
  - successor, 96
- spot::emptiness\_check\_status, 96
  - ~emptiness\_check\_status, 98
  - aut, 98
  - emptiness\_check\_status, 98
  - h, 98
  - print\_stats, 98
  - root, 98
- spot::explicit\_connected\_component, 99
  - ~explicit\_connected\_component, 101
  - condition, 101
  - has\_state, 101
  - index, 101
  - insert, 101
- spot::explicit\_connected\_component\_factory, 101
  - ~explicit\_connected\_component\_factory, 102
  - build, 102
- spot::free\_list, 105
  - ~free\_list, 107
  - dump\_free\_list, 107
  - extend, 107
  - fl, 108
  - free\_list\_type, 106
  - insert, 107
  - pos\_lenght\_pair, 106
  - register\_n, 107
  - release\_n, 107
  - remove, 107
- spot::gspn\_exeption, 108
  - err\_, 108
  - get\_err, 108
  - get\_where, 108
  - gspn\_exeption, 108
  - where\_, 108
- spot::gspn\_interface, 109
  - ~gspn\_interface, 110
  - automaton, 110
  - dead\_, 110
  - dict\_, 110
  - env\_, 110
  - gspn\_interface, 110
- spot::gspn\_ssp\_interface, 110
  - ~gspn\_ssp\_interface, 111
  - automaton, 112
  - dict\_, 112
  - env\_, 112
  - gspn\_ssp\_interface, 111
- spot::ltl, 25
  - basic\_reduce, 28
  - clone, 28
  - destroy, 28
  - dotty, 28
  - dump, 28
  - format\_parse\_errors, 29
  - is\_eventual, 29
  - is\_FG, 29
  - is\_GF, 29
  - is\_universal, 29

- length, 30
- negative\_normal\_form, 30
- parse, 30
- parse\_error, 28
- parse\_error\_list, 28
- reduce, 30
- Reduce\_All, 28
- Reduce\_Basics, 28
- Reduce\_Eventuality\_And\_Universality, 28
- Reduce\_None, 28
- reduce\_options, 28
- Reduce\_Syntactic\_Implications, 28
- simplify\_f\_g, 31
- syntactic\_implication, 31
- syntactic\_implication\_neg, 31
- to\_spin\_string, 31
- to\_string, 31, 32
- unabbreviate\_logic, 32
- unabbreviate\_ltl, 32
- spot::ltl::atomic\_prop, 34
  - ~atomic\_prop, 38
  - accept, 38
  - atomic\_prop, 38
  - dump\_instances, 38
  - env, 38
  - env\_, 39
  - instance, 38
  - instance\_count, 38
  - instances, 39
  - map, 37
  - name, 38
  - name\_, 39
  - pair, 37
  - ref, 38
  - ref\_, 38
  - ref\_count\_, 39
  - unref, 39
  - unref\_, 39
- spot::ltl::binop, 57
  - ~binop, 60
  - accept, 60
  - binop, 60
  - Equiv, 60
  - first, 61
  - first\_, 62
  - Implies, 60
  - instance, 61
  - instance\_count, 61
  - instances, 62
  - map, 60
  - op, 61
  - op\_, 62
  - op\_name, 61
  - pair, 60
  - pairf, 60
  - R, 60
  - ref, 61
  - ref\_, 61
  - ref\_count\_, 61
  - second, 61
  - second\_, 62
  - type, 60
  - U, 60
  - unref, 61
  - unref\_, 62
  - Xor, 60
- spot::ltl::clone\_visitor, 62
  - ~clone\_visitor, 64
  - clone\_visitor, 64
  - recurse, 64
  - result, 64
  - result\_, 65
  - visit, 64, 65
- spot::ltl::const\_visitor, 69
  - visit, 69, 70
- spot::ltl::constant, 70
  - ~constant, 72
  - accept, 72
  - constant, 72
  - False, 72
  - false\_instance, 72
  - ref, 73
  - ref\_, 73
  - True, 72
  - true\_instance, 73
  - type, 72
  - unref, 73
  - unref\_, 73
  - val, 73
  - val\_, 73
  - val\_name, 73
- spot::ltl::declarative\_environment, 76
  - ~declarative\_environment, 78
  - declarative\_environment, 78
  - declare, 78
  - get\_prop\_map, 78
  - name, 78
  - prop\_map, 77
  - props\_, 78
  - require, 78
- spot::ltl::default\_environment, 78
  - ~default\_environment, 80
  - default\_environment, 80
  - instance, 80
  - name, 80
  - require, 80
- spot::ltl::environment, 98
  - ~environment, 99

- name, 99
- require, 99
- spot::ltl::formula, 102
  - ~formula, 104
  - accept, 104
  - ref, 104
  - ref\_, 104
  - unref, 104
  - unref\_, 104
- spot::ltl::multop, 121
  - ~multop, 125
  - accept, 125
  - And, 125
  - children\_, 127
  - instance, 126
  - instance\_count, 126
  - instances, 127
  - map, 125
  - multop, 125
  - nth, 126
  - op, 126
  - op\_, 127
  - op\_name, 126
  - Or, 125
  - pair, 125
  - ref, 126
  - ref\_, 127
  - ref\_count\_, 127
  - size, 127
  - type, 125
  - unref, 127
  - unref\_, 127
  - vec, 125
- spot::ltl::multop::pairemp, 127
  - operator(), 128
- spot::ltl::postfix\_visitor, 156
  - ~postfix\_visitor, 158
  - doit, 158
  - doit\_default, 158
  - postfix\_visitor, 158
  - visit, 158, 159
- spot::ltl::ref\_formula, 159
  - ~ref\_formula, 161
  - accept, 161
  - ref, 162
  - ref\_, 162
  - ref\_count\_, 162
  - ref\_counter\_, 162
  - ref\_formula, 161
  - unref, 162
  - unref\_, 162
- spot::ltl::simplify\_f\_g\_visitor, 165
  - ~simplify\_f\_g\_visitor, 167
  - recurse, 167
  - result, 167
  - result\_, 167
  - simplify\_f\_g\_visitor, 167
  - super, 167
  - visit, 167
- spot::ltl::unabbreviate\_logic\_visitor, 271
  - ~unabbreviate\_logic\_visitor, 273
  - recurse, 273
  - result, 273
  - result\_, 274
  - super, 273
  - unabbreviate\_logic\_visitor, 273
  - visit, 273
- spot::ltl::unabbreviate\_ltl\_visitor, 274
  - ~unabbreviate\_ltl\_visitor, 276
  - recurse, 276
  - result, 276
  - result\_, 277
  - super, 276
  - unabbreviate\_ltl\_visitor, 276
  - visit, 276
- spot::ltl::unop, 277
  - ~unop, 280
  - accept, 280
  - child, 280
  - child\_, 281
  - F, 280
  - G, 280
  - instance, 280
  - instance\_count, 280
  - instances, 281
  - map, 280
  - Not, 280
  - op, 280
  - op\_, 281
  - op\_name, 281
  - pair, 280
  - ref, 281
  - ref\_, 281
  - ref\_count\_, 281
  - type, 280
  - unop, 280
  - unref, 281
  - unref\_, 281
  - X, 280
- spot::ltl::visitor, 282
  - visit, 283
- spot::magic\_search, 114
  - ~magic\_search, 116
  - a, 116
  - check, 116
  - h, 116
  - has, 116
  - hash\_type, 115

- magic\_search, 116
- print\_result, 116
- push, 116
- stack, 116
- stack\_type, 115
- state\_iter\_pair, 116
- tstack, 117
- tstack\_type, 116
- x, 117
- spot::magic\_search::magic, 117
  - seen\_with, 117
  - seen\_without, 117
- spot::magic\_search::magic\_state, 117
  - m, 118
  - s, 118
- spot::minato\_isop, 118
  - cube\_, 120
  - minato\_isop, 119
  - next, 119
  - ret\_, 120
  - todo\_, 120
- spot::minato\_isop::local\_vars
  - FirstStep, 120
  - FourthStep, 121
  - SecondStep, 120
  - ThirdStep, 121
- spot::minato\_isop::local\_vars, 120
  - f0\_max, 121
  - f0\_min, 121
  - f1\_max, 121
  - f1\_min, 121
  - f\_max, 121
  - f\_min, 121
  - g0, 121
  - g1, 121
  - local\_vars, 121
  - step, 121
  - v1, 121
  - vars, 121
- spot::numbered\_state\_heap, 128
  - ~numbered\_state\_heap, 129
  - find, 129
  - index, 130
  - insert, 130
  - iterator, 130
  - size, 130
  - state\_index, 129
  - state\_index\_p, 129
- spot::numbered\_state\_heap\_const\_iterator, 130
  - ~numbered\_state\_heap\_const\_iterator, 131
  - done, 131
  - first, 131
  - get\_index, 131
  - get\_state, 131
  - next, 131
- spot::numbered\_state\_heap\_factory, 131
  - ~numbered\_state\_heap\_factory, 132
  - build, 132
- spot::numbered\_state\_heap\_hash\_map, 132
  - ~numbered\_state\_heap\_hash\_map, 134
  - find, 134
  - h, 135
  - hash\_type, 134
  - index, 135
  - insert, 135
  - iterator, 135
  - size, 135
  - state\_index, 134
  - state\_index\_p, 134
- spot::numbered\_state\_heap\_hash\_map\_factory, 136
  - ~numbered\_state\_heap\_hash\_map\_factory, 137
  - build, 137
  - instance, 137
  - numbered\_state\_heap\_hash\_map\_factory, 137
- spot::parity\_game\_graph, 137
  - ~parity\_game\_graph, 141
  - add\_state, 141
  - automata\_, 142
  - build\_graph, 141
  - duplicator\_vertice\_, 142
  - end, 141
  - get\_relation, 141
  - lift, 141
  - nb\_node\_parity\_game, 142
  - next\_state, 141
  - parity\_game\_graph, 141
  - print, 141
  - process\_link, 141
  - process\_state, 142
  - run, 142
  - seen, 142
  - seen\_map, 140
  - spoiler\_vertice\_, 142
  - start, 142
  - tgba\_state\_, 142
  - todo, 142
- spot::parity\_game\_graph\_delayed, 143
  - ~parity\_game\_graph\_delayed, 146
  - add\_duplicator\_node\_delayed, 146
  - add\_spoiler\_node\_delayed, 146
  - add\_state, 146
  - automata\_, 148
  - bdd\_v, 146
  - build\_graph, 147
  - build\_recurse\_successor\_duplicator, 147



- build\_recurse\_successor\_spoiler, 147
- build\_sub\_set\_acc\_cond, 147
- duplicator\_vertice\_, 148
- end, 147
- get\_relation, 147
- lift, 147
- nb\_node\_parity\_game, 148
- nb\_set\_acc\_cond, 147
- next\_state, 147
- parity\_game\_graph\_delayed, 146
- print, 147
- process\_link, 147
- process\_state, 148
- run, 148
- seen, 148
- seen\_map, 146
- spoiler\_vertice\_, 148
- start, 148
- sub\_set\_acc\_cond\_, 149
- tgba\_state\_, 149
- todo, 149
- spot::parity\_game\_graph\_direct, 149
  - ~parity\_game\_graph\_direct, 152
  - add\_state, 153
  - automata\_, 154
  - build\_graph, 153
  - build\_link, 153
  - duplicator\_vertice\_, 154
  - end, 153
  - get\_relation, 153
  - lift, 153
  - nb\_node\_parity\_game, 154
  - next\_state, 153
  - parity\_game\_graph\_direct, 152
  - print, 153
  - process\_link, 153
  - process\_state, 154
  - run, 154
  - seen, 154
  - seen\_map, 152
  - spoiler\_vertice\_, 154
  - start, 154
  - tgba\_state\_, 154
  - todo, 154
- spot::ptr\_hash, 159
  - operator(), 159
- spot::scc\_stack, 162
  - empty, 163
  - pop, 163
  - push, 163
  - s, 164
  - size, 163
  - stack\_type, 163
  - top, 163
- spot::scc\_stack::connected\_component, 164
  - condition, 164
  - connected\_component, 164
  - index, 165
- spot::spoiler\_node, 169
  - ~spoiler\_node, 170
  - add\_pred, 170
  - add\_succ, 170
  - compare, 170
  - del\_pred, 170
  - del\_succ, 170
  - get\_duplicator\_node, 170
  - get\_nb\_succ, 171
  - get\_pair, 171
  - get\_spoiler\_node, 171
  - lnode\_pred, 171
  - lnode\_succ, 171
  - not\_win, 171
  - num\_, 171
  - prune, 171
  - sc\_, 171
  - set\_win, 171
  - spoiler\_node, 170
  - succ\_to\_string, 171
  - to\_string, 171
- spot::spoiler\_node\_delayed, 171
  - ~spoiler\_node\_delayed, 173
  - acceptance\_condition\_visited\_, 175
  - add\_pred, 174
  - add\_succ, 174
  - compare, 174
  - del\_pred, 174
  - del\_succ, 174
  - get\_acceptance\_condition\_visited, 174
  - get\_duplicator\_node, 174
  - get\_lead\_2\_acc\_all, 174
  - get\_nb\_succ, 174
  - get\_pair, 174
  - get\_progress\_measure, 174
  - get\_spoiler\_node, 174
  - lead\_2\_acc\_all\_, 175
  - lnode\_pred, 175
  - lnode\_succ, 175
  - not\_win, 175
  - num\_, 175
  - progress\_measure\_, 175
  - prune, 174
  - sc\_, 175
  - set\_win, 174
  - spoiler\_node\_delayed, 173
  - succ\_to\_string, 174
  - to\_string, 174
- spot::state, 177
  - ~state, 178

- clone, 178
- compare, 178
- hash, 178
- spot::state\_bdd, 179
  - as\_bdd, 180
  - clone, 180
  - compare, 180
  - hash, 180
  - state\_, 181
  - state\_bdd, 180
- spot::state\_explicit, 181
  - ~state\_explicit, 182
  - clone, 183
  - compare, 183
  - get\_state, 183
  - hash, 183
  - state\_, 183
  - state\_explicit, 182
- spot::state\_product, 184
  - ~state\_product, 185
  - clone, 186
  - compare, 186
  - hash, 186
  - left, 186
  - left\_, 186
  - right, 186
  - right\_, 186
  - state\_product, 185
- spot::state\_ptr\_equal, 187
  - operator(), 187
- spot::state\_ptr\_hash, 187
  - operator(), 188
- spot::state\_ptr\_less\_than, 188
  - operator(), 188
- spot::string\_hash, 188
  - operator(), 189
- spot::tgba, 189
  - ~tgba, 191
  - all\_acceptance\_conditions, 191
  - compute\_support\_conditions, 191
  - compute\_support\_variables, 192
  - format\_state, 192
  - get\_dict, 192
  - get\_init\_state, 192
  - last\_support\_conditions\_input\_, 194
  - last\_support\_conditions\_output\_, 194
  - last\_support\_variables\_input\_, 194
  - last\_support\_variables\_output\_, 194
  - neg\_acceptance\_conditions, 192
  - project\_state, 192
  - succ\_iter, 193
  - support\_conditions, 193
  - support\_variables, 193
  - tgba, 191
- spot::tgba\_bdd\_concrete, 194
  - ~tgba\_bdd\_concrete, 197
  - all\_acceptance\_conditions, 197
  - compute\_support\_conditions, 197
  - compute\_support\_variables, 197
  - data\_, 200
  - format\_state, 197
  - get\_core\_data, 197
  - get\_dict, 198
  - get\_init\_bdd, 198
  - get\_init\_state, 198
  - init\_, 200
  - neg\_acceptance\_conditions, 198
  - project\_state, 198
  - set\_init\_state, 199
  - succ\_iter, 199
  - support\_conditions, 199
  - support\_variables, 199
  - tgba\_bdd\_concrete, 197
  - tgba\_bdd\_concrete::operator=, 199
- spot::tgba\_bdd\_concrete\_factory, 200
  - ~tgba\_bdd\_concrete\_factory, 202
  - acc\_, 203
  - acc\_map\_, 202
  - constrain\_relation, 202
  - create\_atomic\_prop, 202
  - create\_state, 202
  - data\_, 203
  - declare\_acceptance\_condition, 203
  - finish, 203
  - get\_core\_data, 203
  - get\_dict, 203
  - tgba\_bdd\_concrete\_factory, 202
- spot::tgba\_bdd\_core\_data, 203
  - acc\_set, 206
  - acceptance\_conditions, 206
  - all\_acceptance\_conditions, 207
  - declare\_acceptance\_condition, 206
  - declare\_atomic\_prop, 206
  - declare\_now\_next, 206
  - dict, 207
  - negacc\_set, 207
  - next\_set, 207
  - notacc\_set, 207
  - notnext\_set, 207
  - notnow\_set, 207
  - notvar\_set, 208
  - now\_set, 208
  - nownext\_set, 208
  - operator=, 206
  - relation, 208
  - tgba\_bdd\_core\_data, 206
  - var\_set, 208
  - varandnext\_set, 208

- spot::tgba\_bdd\_factory, 208
  - get\_core\_data, 209
- spot::tgba\_explicit, 209
  - ~tgba\_explicit, 213
  - add\_acceptance\_condition, 214
  - add\_acceptance\_conditions, 214
  - add\_condition, 214
  - add\_conditions, 214
  - add\_state, 214
  - all\_acceptance\_conditions, 214
  - all\_acceptance\_conditions\_, 217
  - all\_acceptance\_conditions\_computed\_, 217
  - complement\_all\_acceptance\_conditions, 214
  - compute\_support\_conditions, 214
  - compute\_support\_variables, 214
  - create\_transition, 214
  - declare\_acceptance\_condition, 215
  - dict\_, 217
  - format\_state, 215
  - get\_acceptance\_condition, 215
  - get\_dict, 215
  - get\_init\_state, 215
  - has\_acceptance\_condition, 215
  - init\_, 217
  - merge\_transitions, 215
  - name\_state\_map\_, 217
  - neg\_acceptance\_conditions, 215
  - neg\_acceptance\_conditions\_, 217
  - ns\_map, 213
  - project\_state, 216
  - set\_init\_state, 216
  - sn\_map, 213
  - state, 213
  - state\_name\_map\_, 217
  - succ\_iter, 216
  - support\_conditions, 216
  - support\_variables, 217
  - tgba\_explicit, 213
  - tgba\_explicit::operator=, 217
- spot::tgba\_explicit::transition, 217
  - acceptance\_conditions, 218
  - condition, 218
  - dest, 218
- spot::tgba\_explicit\_succ\_iterator, 218
  - ~tgba\_explicit\_succ\_iterator, 220
  - all\_acceptance\_conditions\_, 221
  - current\_acceptance\_conditions, 220
  - current\_condition, 220
  - current\_state, 220
  - done, 220
  - first, 221
  - i\_, 221
  - next, 221
  - s\_, 221
  - tgba\_explicit\_succ\_iterator, 220
- spot::tgba\_product, 221
  - ~tgba\_product, 225
  - all\_acceptance\_conditions, 225
  - all\_acceptance\_conditions\_, 227
  - compute\_support\_conditions, 225
  - compute\_support\_variables, 225
  - dict\_, 227
  - format\_state, 225
  - get\_dict, 225
  - get\_init\_state, 225
  - left\_, 227
  - left\_acc\_complement\_, 227
  - neg\_acceptance\_conditions, 226
  - neg\_acceptance\_conditions\_, 227
  - project\_state, 226
  - right\_, 227
  - right\_acc\_complement\_, 227
  - succ\_iter, 226
  - support\_conditions, 226
  - support\_variables, 227
  - tgba\_product, 224, 225
  - tgba\_product::operator=, 227
- spot::tgba\_reachable\_iterator, 228
  - ~tgba\_reachable\_iterator, 230
  - add\_state, 230
  - automata\_, 231
  - end, 230
  - next\_state, 230
  - process\_link, 231
  - process\_state, 231
  - run, 231
  - seen, 231
  - seen\_map, 230
  - start, 231
  - tgba\_reachable\_iterator, 230
- spot::tgba\_reachable\_iterator\_breadth\_first, 232
  - add\_state, 234
  - automata\_, 235
  - end, 234
  - next\_state, 234
  - process\_link, 235
  - process\_state, 235
  - run, 235
  - seen, 235
  - seen\_map, 234
  - start, 235
  - tgba\_reachable\_iterator\_breadth\_first, 234
  - todo, 235
- spot::tgba\_reachable\_iterator\_depth\_first, 236
  - add\_state, 238
  - automata\_, 239
  - end, 238

- next\_state, 238
- process\_link, 239
- process\_state, 239
- run, 239
- seen, 239
- seen\_map, 238
- start, 239
- tgba\_reachable\_iterator\_depth\_first, 238
- todo, 239
- spot::tgba\_reduc, 240
  - ~tgba\_reduc, 246
  - acc\_, 252
  - add\_acceptance\_condition, 246
  - add\_acceptance\_conditions, 246
  - add\_condition, 246
  - add\_conditions, 246
  - add\_state, 246
  - all\_acceptance\_conditions, 247
  - all\_acceptance\_conditions\_, 252
  - all\_acceptance\_conditions\_computed\_, 253
  - automata\_, 253
  - complement\_all\_acceptance\_conditions, 247
  - compute\_scc, 247
  - compute\_support\_conditions, 247
  - compute\_support\_variables, 247
  - create\_transition, 247
  - declare\_acceptance\_condition, 247
  - delete\_scc, 247
  - dict\_, 253
  - display\_rel\_sim, 248
  - display\_scc, 248
  - end, 248
  - format\_state, 248
  - get\_acceptance\_condition, 248
  - get\_dict, 248
  - get\_init\_state, 248
  - get\_nb\_state, 248
  - get\_nb\_transition, 248
  - h\_, 253
  - has\_acceptance\_condition, 249
  - init\_, 253
  - is\_alpha\_ball, 249
  - is\_not\_accepting, 249
  - is\_terminal, 249
  - merge\_state, 249
  - merge\_transitions, 249
  - name\_state\_map\_, 253
  - neg\_acceptance\_conditions, 249
  - neg\_acceptance\_conditions\_, 253
  - next\_state, 249
  - ns\_map, 246
  - process\_link, 250
  - process\_state, 250
  - project\_state, 250
  - prune\_acc, 250
  - prune\_automata, 250
  - prune\_scc, 250
  - quotient\_state, 251
  - redirect\_transition, 251
  - remove\_acc, 251
  - remove\_component, 251
  - remove\_predecessor\_state, 251
  - remove\_scc, 251
  - remove\_scc\_depth\_first, 251
  - remove\_state, 251
  - root\_, 253
  - run, 251
  - scc\_computed\_, 253
  - seen, 253
  - seen\_, 253
  - seen\_map, 246
  - set\_init\_state, 251
  - si\_, 253
  - sn\_map, 246
  - sp\_map, 246
  - start, 251
  - state, 246
  - state\_name\_map\_, 253
  - state\_predecessor\_map\_, 253
  - state\_scc\_, 253
  - state\_scc\_v\_, 253
  - succ\_iter, 251, 252
  - support\_conditions, 252
  - support\_variables, 252
  - tgba\_reduc, 246
  - todo, 253
- spot::tgba\_statistics, 254
  - states, 254
  - transitions, 254
- spot::tgba\_succ\_iterator, 254
  - ~tgba\_succ\_iterator, 255
  - current\_acceptance\_conditions, 255
  - current\_condition, 255
  - current\_state, 256
  - done, 256
  - first, 256
  - next, 256
- spot::tgba\_succ\_iterator\_concrete, 257
  - ~tgba\_succ\_iterator\_concrete, 259
  - current\_, 260
  - current\_acc\_, 260
  - current\_acceptance\_conditions, 259
  - current\_condition, 259
  - current\_state, 260
  - current\_state\_, 261
  - data\_, 261
  - done, 260

- first, 260
- next, 260
- succ\_set\_, 261
- succ\_set\_left\_, 261
- tgba\_succ\_iterator\_concrete, 259
- spot::tgba\_succ\_iterator\_product, 261
  - ~tgba\_succ\_iterator\_product, 263
  - current\_acceptance\_conditions, 263
  - current\_cond\_, 264
  - current\_condition, 263
  - current\_state, 263
  - done, 263
  - first, 264
  - left\_, 264
  - left\_neg\_, 264
  - next, 264
  - next\_non\_false\_, 264
  - right\_, 264
  - right\_neg\_, 264
  - step\_, 264
  - tgba\_succ\_iterator\_product, 263
- spot::tgba\_tba\_proxy, 265
  - ~tgba\_tba\_proxy, 268
  - a\_, 270
  - acc\_cycle\_, 270
  - all\_acceptance\_conditions, 268
  - compute\_support\_conditions, 268
  - compute\_support\_variables, 268
  - cycle\_list, 268
  - format\_state, 268
  - get\_dict, 268
  - get\_init\_state, 269
  - neg\_acceptance\_conditions, 269
  - project\_state, 269
  - state\_is\_accepting, 269
  - succ\_iter, 269
  - support\_conditions, 270
  - support\_variables, 270
  - tgba\_tba\_proxy, 268
  - tgba\_tba\_proxy::operator=, 270
  - the\_acceptance\_cond\_, 270
- Stack
  - yy::Stack, 176
- stack
  - spot::magic\_search, 116
- stack\_
  - yy::Slice, 168
- stack\_type
  - spot::magic\_search, 115
  - spot::scc\_stack, 163
- start
  - spot::parity\_game\_graph, 142
  - spot::parity\_game\_graph\_delayed, 148
  - spot::parity\_game\_graph\_direct, 154
  - spot::tgba\_reachable\_iterator, 231
  - spot::tgba\_reachable\_iterator\_breadth\_first, 235
  - spot::tgba\_reachable\_iterator\_depth\_first, 239
  - spot::tgba\_reduc, 251
- state
  - spot::tgba\_explicit, 213
  - spot::tgba\_reduc, 246
- state\_
  - spot::state\_bdd, 181
  - spot::state\_explicit, 183
- state\_bdd
  - spot::state\_bdd, 180
- state\_couple
  - spot, 17
- state\_explicit
  - spot::state\_explicit, 182
- state\_index
  - spot::numbered\_state\_heap, 129
  - spot::numbered\_state\_heap\_hash\_map, 134
- state\_index\_p
  - spot::numbered\_state\_heap, 129
  - spot::numbered\_state\_heap\_hash\_map, 134
- state\_is\_accepting
  - spot::tgba\_tba\_proxy, 269
- state\_iter\_pair
  - spot::magic\_search, 116
- state\_name\_map\_
  - spot::tgba\_explicit, 217
  - spot::tgba\_reduc, 253
- state\_predecessor\_map\_
  - spot::tgba\_reduc, 253
- state\_product
  - spot::state\_product, 185
- state\_proposition
  - spot::counter\_example, 75
- state\_scc\_
  - spot::tgba\_reduc, 253
- state\_scc\_v\_
  - spot::tgba\_reduc, 253
- state\_sequence
  - spot::counter\_example, 75
- states
  - spot::connected\_component\_hash\_set, 67
  - spot::tgba\_statistics, 254
- stats\_reachable
  - spot, 23
- step
  - spot::minato\_isop::local\_vars, 121
  - yy::Location, 113
- step\_
  - spot::tgba\_succ\_iterator\_product, 264
- sub\_set\_acc\_cond\_

- spot::parity\_game\_graph\_delayed, 149
- succ\_iter
  - spot::tgba, 193
  - spot::tgba\_bdd\_concrete, 199
  - spot::tgba\_explicit, 216
  - spot::tgba\_product, 226
  - spot::tgba\_reduc, 251, 252
  - spot::tgba\_tba\_proxy, 269
- succ\_queue
  - spot::emptiness\_check\_shy, 94
- succ\_set\_
  - spot::tgba\_succ\_iterator\_concrete, 261
- succ\_set\_left\_
  - spot::tgba\_succ\_iterator\_concrete, 261
- succ\_to\_string
  - spot::duplicator\_node, 84
  - spot::duplicator\_node\_delayed, 88
  - spot::spoiler\_node, 171
  - spot::spoiler\_node\_delayed, 174
- successor
  - spot::emptiness\_check\_shy::successor, 96
- suffix
  - spot::counter\_example, 76
- super
  - spot::ltl::simplify\_f\_g\_visitor, 167
  - spot::ltl::unabbreviate\_logic\_visitor, 273
  - spot::ltl::unabbreviate\_ltl\_visitor, 276
- support\_conditions
  - spot::tgba, 193
  - spot::tgba\_bdd\_concrete, 199
  - spot::tgba\_explicit, 216
  - spot::tgba\_product, 226
  - spot::tgba\_reduc, 252
  - spot::tgba\_tba\_proxy, 270
- support\_variables
  - spot::tgba, 193
  - spot::tgba\_bdd\_concrete, 199
  - spot::tgba\_explicit, 217
  - spot::tgba\_product, 227
  - spot::tgba\_reduc, 252
  - spot::tgba\_tba\_proxy, 270
- syntactic\_implication
  - spot::ltl, 31
- syntactic\_implication\_neg
  - spot::ltl, 31
- tgba
  - spot::tgba, 191
- tgba/bdddict.hh, 309
- tgba/bddprint.hh, 310
- tgba/formula2bdd.hh, 310
- tgba/public.hh, 296
- tgba/state.hh, 311
- tgba/statebdd.hh, 312
- tgba/succiter.hh, 312
- tgba/succiterconcrete.hh, 313
- tgba/tgba.hh, 314
- tgba/tgababddconcrete.hh, 314
- tgba/tgababddconcretefactory.hh, 315
- tgba/tgababddconcreteproduct.hh, 316
- tgba/tgababddcoredata.hh, 317
- tgba/tgababddfactory.hh, 318
- tgba/tgbaexplicit.hh, 318
- tgba/tgabaproduct.hh, 319
- tgba/tgabareduc.hh, 320
- tgba/tgbatba.hh, 320
- tgba\_bdd\_concrete
  - spot::tgba\_bdd\_concrete, 197
- tgba\_bdd\_concrete::operator=
  - spot::tgba\_bdd\_concrete, 199
- tgba\_bdd\_concrete\_factory
  - spot::tgba\_bdd\_concrete\_factory, 202
- tgba\_bdd\_core\_data
  - spot::tgba\_bdd\_core\_data, 206
- tgba\_dupexp\_bfs
  - spot, 24
- tgba\_dupexp\_dfs
  - spot, 24
- tgba\_explicit
  - spot::tgba\_explicit, 213
- tgba\_explicit::operator=
  - spot::tgba\_explicit, 217
- tgba\_explicit\_succ\_iterator
  - spot::tgba\_explicit\_succ\_iterator, 220
- tgba\_parse
  - spot, 24
- tgba\_parse\_error
  - spot, 17
- tgba\_parse\_error\_list
  - spot, 17
- tgba\_powerset
  - spot, 24
- tgba\_product
  - spot::tgba\_product, 224, 225
- tgba\_product::operator=
  - spot::tgba\_product, 227
- tgba\_reachable\_iterator
  - spot::tgba\_reachable\_iterator, 230
- tgba\_reachable\_iterator\_breadth\_first
  - spot::tgba\_reachable\_iterator\_breadth\_first, 234
- tgba\_reachable\_iterator\_depth\_first
  - spot::tgba\_reachable\_iterator\_depth\_first, 238
- tgba\_reduc
  - spot::tgba\_reduc, 246
- tgba\_save\_reachable
  - spot, 24

- tgba\_state\_
  - spot::parity\_game\_graph, 142
  - spot::parity\_game\_graph\_delayed, 149
  - spot::parity\_game\_graph\_direct, 154
- tgba\_succ\_iterator\_concrete
  - spot::tgba\_succ\_iterator\_concrete, 259
- tgba\_succ\_iterator\_product
  - spot::tgba\_succ\_iterator\_product, 263
- tgba\_tba\_proxy
  - spot::tgba\_tba\_proxy, 268
- tgba\_tba\_proxy::operator=
  - spot::tgba\_tba\_proxy, 270
- tgbaalgos/dotty.hh, 300
- tgbaalgos/duexp.hh, 321
- tgbaalgos/gtec/ce.hh, 322
- tgbaalgos/gtec/explsc.hh, 323
- tgbaalgos/gtec/gtec.hh, 323
- tgbaalgos/gtec/nsheap.hh, 324
- tgbaalgos/gtec/scstack.hh, 325
- tgbaalgos/gtec/status.hh, 326
- tgbaalgos/lbtt.hh, 326
- tgbaalgos/ltl2tgba\_fm.hh, 327
- tgbaalgos/ltl2tgba\_lacim.hh, 328
- tgbaalgos/magic.hh, 328
- tgbaalgos/neverclaim.hh, 329
- tgbaalgos/powerset.hh, 330
- tgbaalgos/reachiter.hh, 330
- tgbaalgos/reductgba\_sim.hh, 331
- tgbaalgos/save.hh, 332
- tgbaalgos/stats.hh, 332
- tgbaparse/public.hh, 296
- the\_acceptance\_cond\_
  - spot::tgba\_tba\_proxy, 270
- ThirdStep
  - spot::minato\_isop::local\_vars, 121
- to\_spin\_string
  - spot::ltl, 31
- to\_string
  - spot::duplicator\_node, 84
  - spot::duplicator\_node\_delayed, 89
  - spot::ltl, 31, 32
  - spot::spoiler\_node, 171
  - spot::spoiler\_node\_delayed, 174
- todo
  - spot::emptiness\_check\_shy, 95
  - spot::parity\_game\_graph, 142
  - spot::parity\_game\_graph\_delayed, 149
  - spot::parity\_game\_graph\_direct, 154
  - spot::tgba\_reachable\_iterator\_breadth\_first, 235
  - spot::tgba\_reachable\_iterator\_depth\_first, 239
  - spot::tgba\_reduc, 253
- todo\_
  - spot::minato\_isop, 120
- top
  - spot::scc\_stack, 163
- transitions
  - spot::tgba\_statistics, 254
- True
  - spot::ltl::constant, 72
- true\_instance
  - spot::ltl::constant, 73
- tstack
  - spot::magic\_search, 117
- tstack\_type
  - spot::magic\_search, 116
- type
  - spot::ltl::binop, 60
  - spot::ltl::constant, 72
  - spot::ltl::multop, 125
  - spot::ltl::unop, 280
- U
  - spot::ltl::binop, 60
- unabbreviate\_logic
  - spot::ltl, 32
- unabbreviate\_logic\_visitor
  - spot::ltl::unabbreviate\_logic\_visitor, 273
- unabbreviate\_ltl
  - spot::ltl, 32
- unabbreviate\_ltl\_visitor
  - spot::ltl::unabbreviate\_ltl\_visitor, 276
- unop
  - spot::ltl::unop, 280
- unref
  - spot::ltl::atomic\_prop, 39
  - spot::ltl::binop, 61
  - spot::ltl::constant, 73
  - spot::ltl::formula, 104
  - spot::ltl::multop, 127
  - spot::ltl::ref\_formula, 162
  - spot::ltl::unop, 281
- unref\_
  - spot::ltl::atomic\_prop, 39
  - spot::ltl::binop, 62
  - spot::ltl::constant, 73
  - spot::ltl::formula, 104
  - spot::ltl::multop, 127
  - spot::ltl::ref\_formula, 162
  - spot::ltl::unop, 281
- unregister\_all\_my\_variables
  - spot::bdd\_dict, 52
- unregister\_variable
  - spot::bdd\_dict, 52
- v1
  - spot::minato\_isop::local\_vars, 121

- val
  - spot::ltl::constant, 73
- val\_
  - spot::ltl::constant, 73
- val\_name
  - spot::ltl::constant, 73
- var\_formula\_map
  - spot::bdd\_dict, 53
- var\_map
  - spot::bdd\_dict, 53
- var\_refs
  - spot::bdd\_dict, 53
- var\_set
  - spot::tgba\_bdd\_core\_data, 208
- varandnext\_set
  - spot::tgba\_bdd\_core\_data, 208
- vars
  - spot::minato\_isop::local\_vars, 121
- vec
  - spot::ltl::multop, 125
- version
  - spot, 24
- vf\_map
  - spot::bdd\_dict, 49
- visit
  - spot::ltl::clone\_visitor, 64, 65
  - spot::ltl::const\_visitor, 69, 70
  - spot::ltl::postfix\_visitor, 158, 159
  - spot::ltl::simplify\_f\_g\_visitor, 167
  - spot::ltl::unabbreviate\_logic\_visitor, 273
  - spot::ltl::unabbreviate\_ltl\_visitor, 276
  - spot::ltl::visitor, 283
- vr\_map
  - spot::bdd\_dict, 49
- where\_
  - spot::gspn\_exeption, 108
- X
  - spot::ltl::unop, 280
- x
  - spot::magic\_search, 117
- Xor
  - spot::ltl::binop, 60
- yy, 32
  - operator+, 33
  - operator+=", 33
  - operator-, 33
  - operator=, 33
  - operator<<, 33, 34
- yy::Location, 112
  - begin, 113
  - columns, 113
  - end, 113
  - lines, 113
  - Location, 113
  - step, 113
- yy::Position, 155
  - column, 156
  - columns, 156
  - filename, 156
  - initial\_column, 156
  - initial\_line, 156
  - line, 156
  - lines, 156
  - Position, 156
- yy::Slice, 168
  - operator[], 168
  - range\_, 168
  - Slice, 168
  - stack\_, 168
- yy::Stack, 175
  - begin, 176
  - ConstIterator, 176
  - end, 176
  - height, 176
  - Iterator, 176
  - operator[], 176, 177
  - pop, 177
  - push, 177
  - seq\_, 177
  - Stack, 176