

spot Reference Manual
0.01

Generated by Doxygen 1.3.4

Mon Dec 1 11:33:53 2003

Contents

1 spot Main Page	1
2 spot Namespace Index	1
3 spot Hierarchical Index	2
4 spot Class Index	4
5 spot File Index	6
6 spot Namespace Documentation	8
7 spot Class Documentation	23
8 spot File Documentation	155

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](#) Convert a [spot::ltl::formula](#) into a [spot::tgba](#).

2 spot Namespace Index

2.1 spot Namespace List

Here is a list of all namespaces with brief descriptions:

spot	8
spot::ltl	16
yy	21

3 spot Hierarchical Index

3.1 spot Class Hierarchy

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

spot::bdd_allocator	26
spot::bdd_dict	28
spot::bdd_less_than	35
spot::ltl::const_visitor	42
spot::emptiness_check	48
spot::emptiness_check::connected_component	51
spot::emptiness_check::connected_component_set	51
spot::ltl::environment	53
spot::ltl::default_environment	46
spot::ltl::formula	54
spot::ltl::constant	43
spot::ltl::ref_formula	74
spot::ltl::atomic_prop	23
spot::ltl::binop	36
spot::ltl::multop	65
spot::ltl::unop	150
yy::Location	56
spot::magic_search	57
spot::magic_search::magic	61
spot::magic_search::magic_state	61
spot::minato_isop	62
spot::minato_isop::local_vars	63
spot::ltl::multop::pairemp	70
yy::Position	70
spot::ptr_hash< T >	74
yy::Slice< T, S >	77

<code>yy::Stack< T, S ></code>	78
<code>spot::state</code>	79
<code>spot::state_bdd</code>	81
<code>spot::state_explicit</code>	83
<code>spot::state_product</code>	85
<code>spot::state_ptr_equal</code>	87
<code>spot::state_ptr_hash</code>	88
<code>spot::state_ptr_less_than</code>	89
<code>spot::string_hash</code>	89
<code>spot::tgba</code>	90
<code>spot::tgba_bdd_concrete</code>	94
<code>spot::tgba_explicit</code>	108
<code>spot::tgba_product</code>	117
<code>spot::tgba_tba_proxy</code>	140
<code>spot::tgba_bdd_core_data</code>	102
<code>spot::tgba_bdd_factory</code>	107
<code>spot::tgba_bdd_concrete_factory</code>	99
<code>spot::tgba_explicit::transition</code>	114
<code>spot::tgba_reachable_iterator</code>	122
<code>spot::tgba_reachable_iterator_breadth_first</code>	125
<code>spot::tgba_reachable_iterator_depth_first</code>	128
<code>spot::tgba_succ_iterator</code>	131
<code>spot::tgba_explicit_succ_iterator</code>	115
<code>spot::tgba_succ_iterator_concrete</code>	133
<code>spot::tgba_succ_iterator_product</code>	137
<code>spot::ltl::visitor</code>	153
<code>spot::ltl::clone_visitor</code>	40
<code>spot::ltl::unabbreviate_logic_visitor</code>	145
<code>spot::ltl::unabbreviate_ltl_visitor</code>	147

[spot::ltl::postfix_visitor](#) 72

4 spot Class Index

4.1 spot Class List

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

spot::ltl::atomic_prop (Atomic propositions)	23
spot::bdd_allocator (Manage ranges of variables)	26
spot::bdd_dict (Map BDD variables to formulae)	28
spot::bdd_less_than (Comparison functor for BDDs)	35
spot::ltl::binop (Binary operator)	36
spot::ltl::clone_visitor (Clone a formula)	40
spot::ltl::const_visitor (Formula visitor that cannot modify the formula)	42
spot::ltl::constant (A constant (True or False))	43
spot::ltl::default_environment (A laxist environment)	46
spot::emptiness_check (Check whether the language of an automate is empty)	48
spot::emptiness_check::connected_component	51
spot::emptiness_check::connected_component_set	51
spot::ltl::environment (An environment that describe atomic propositions)	53
spot::ltl::formula (An LTL formula)	54
yy::Location (Abstract a Location)	56
spot::magic_search (Emptiness check on spot::tgba_tba_proxy automata using the Magic Search algorithm)	57
spot::magic_search::magic (Records whether a state has be seen with the magic bit on or off)	61
spot::magic_search::magic_state (A state for the spot::magic_search algorithm)	61
spot::minato_isop (Generate an irredundant sum-of-products (ISOP) form of a BDD function)	62
spot::minato_isop::local_vars (Internal variables for minato_isop)	63
spot::ltl::multop (Multi-operand operators)	65
spot::ltl::multop::pairemp (Comparison functor used internally by ltl::multop)	70
yy::Position (Abstract a Position)	70

spot::ltl::postfix_visitor (Apply an algorithm on each node of an AST, during a postfix traversal)	72
spot::ptr_hash< T > (A hash function for pointers)	74
spot::ltl::ref_formula (A reference-counted LTL formula)	74
yy::Slice< T, S >	77
yy::Stack< T, S >	78
spot::state (Abstract class for states)	79
spot::state_bdd (A state whose representation is a BDD)	81
spot::state_explicit (States used by spot::tgba_explicit)	83
spot::state_product (A state for spot::tgba_product)	85
spot::state_ptr_equal (An Equivalence Relation for <code>state*</code>)	87
spot::state_ptr_hash (Hash Function for <code>state*</code>)	88
spot::state_ptr_less_than (Strict Weak Ordering for <code>state*</code>)	89
spot::string_hash (A hash function for strings)	89
spot::tgba (A Transition-based Generalized Büchi Automaton)	90
spot::tgba_bdd_concrete (A concrete spot::tgba implemented using BDDs)	94
spot::tgba_bdd_concrete_factory (Helper class to build a spot::tgba_bdd_concrete object)	99
spot::tgba_bdd_core_data (Core data for a TGBA encoded using BDDs)	102
spot::tgba_bdd_factory (Abstract class for spot::tgba_bdd_concrete factories)	107
spot::tgba_explicit (Explicit representation of a spot::tgba)	108
spot::tgba_explicit::transition (Explicit transitions (used by spot::tgba_explicit))	114
spot::tgba_explicit_succ_iterator (Successor iterators used by spot::tgba_explicit)	115
spot::tgba_product (A lazy product. (States are computed on the fly.))	117
spot::tgba_reachable_iterator (Iterate over all reachable states of a spot::tgba)	122
spot::tgba_reachable_iterator_breadth_first (An implementation of spot::tgba_reachable_iterator that browses states breadth first)	125
spot::tgba_reachable_iterator_depth_first (An implementation of spot::tgba_reachable_iterator that browses states depth first)	128
spot::tgba_succ_iterator (Iterate over the successors of a state)	131
spot::tgba_succ_iterator_concrete (A concrete iterator over successors of a TGBA state)	133

<code>spot::tgba_succ_iterator_product</code> (Iterate over the successors of a product computed on the fly)	137
<code>spot::tgba_tba_proxy</code> (Degeneralize a <code>spot::tgba</code> on the fly)	140
<code>spot::ltl::unabbreviate_logic_visitor</code> (Clone and rewrite a formula to remove most of the abbreviated logical operators)	145
<code>spot::ltl::unabbreviate_ltl_visitor</code> (Clone and rewrite a formula to remove most of the abbreviated LTL and logical operators)	147
<code>spot::ltl::unop</code> (Unary operator)	150
<code>spot::ltl::visitor</code> (Formula visitor that can modify the formula)	153

5 spot File Index

5.1 spot File List

Here is a list of all files with brief descriptions:

<code>allnodes.hh</code>	155
<code>atomic_prop.hh</code>	155
<code>bddalloc.hh</code>	156
<code>bdddict.hh</code>	156
<code>bddlt.hh</code>	157
<code>bddprint.hh</code>	158
<code>binop.hh</code>	158
<code>clone.hh</code>	159
<code>constant.hh</code>	160
<code>defaultenv.hh</code>	160
<code>destroy.hh</code>	161
<code>ltlvisit/dotty.hh</code>	161
<code>tgbaalgos/dotty.hh</code>	162
<code>dump.hh</code>	162
<code>dupexp.hh</code>	163
<code>emptinesscheck.hh</code>	163
<code>environment.hh</code>	164
<code>formula.hh</code>	164

formula2bdd.hh	165
hash.hh	166
lbtt.hh	166
location.hh	167
ltl2tgba_fm.hh	168
ltl2tgba_lacim.hh	168
lunabbrev.hh	169
magic.hh	170
mainpage.dox	170
minato.hh	170
multop.hh	171
nenofrm.hh	171
position.hh	172
postfix.hh	172
predecl.hh	173
ltlparse/public.hh	173
tgba/public.hh	174
tgbaparse/public.hh	175
reachiter.hh	175
reformula.hh	176
save.hh	176
stack.hh	177
state.hh	177
statebdd.hh	178
succiter.hh	179
succiterconcrete.hh	179
tgba.hh	180
tgbabddconcrete.hh	181
tgbabddconcretfactory.hh	182

tgbabddconcreteproduct.hh	182
tgbabddcoredata.hh	183
tgbabddfacy.h	184
tgbexplicit.h	185
tgbaproduct.h	185
tgbatba.h	186
tostring.h	187
tunabbrev.h	187
unop.h	187
version.h	188
visitor.h	188

6 spot Namespace Documentation

6.1 spot Namespace Reference

Classes

- class [bdd_allocator](#)
Manage ranges of variables.
- class [bdd_dict](#)
Map BDD variables to formulae.
- struct [bdd_less_than](#)
Comparison functor for BDDs.
- class [emptiness_check](#)
Check whether the language of an automate is empty.
- struct [emptiness_check::connected_component](#)
- struct [emptiness_check::connected_component_set](#)
- struct [magic_search](#)
Emptiness check on `spot::tgba_tba_proxy` automata using the Magic Search algorithm.
- struct [magic_search::magic](#)
Records whether a state has be seen with the magic bit on or off.
- struct [magic_search::magic_state](#)
A state for the `spot::magic_search` algorithm.
- class [minato_isop](#)

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

- struct [minato_isop::local_vars](#)
Internal variables for [minato_isop](#).
- struct [ptr_hash](#)
A hash function for pointers.
- class [state](#)
Abstract class for states.
- class [state_bdd](#)
A state whose representation is a BDD.
- class [state_explicit](#)
States used by [spot::tgba_explicit](#).
- class [state_product](#)
A state for [spot::tgba_product](#).
- struct [state_ptr_equal](#)
An Equivalence Relation for `state`.*
- struct [state_ptr_hash](#)
Hash Function for `state`.*
- struct [state_ptr_less_than](#)
Strict Weak Ordering for `state`.*
- struct [string_hash](#)
A hash function for strings.
- class [tgba](#)
A Transition-based Generalized Büchi Automaton.
- class [tgba_bdd_concrete](#)
A concrete [spot::tgba](#) implemented using BDDs.
- class [tgba_bdd_concrete_factory](#)
Helper class to build a [spot::tgba_bdd_concrete](#) object.
- struct [tgba_bdd_core_data](#)
Core data for a TGBA encoded using BDDs.
- class [tgba_bdd_factory](#)
Abstract class for [spot::tgba_bdd_concrete](#) factories.
- class [tgba_explicit](#)
Explicit representation of a [spot::tgba](#).

- struct [tgba_explicit::transition](#)
Explicit transitions (used by [spot::tgba_explicit](#)).
- class [tgba_explicit_succ_iterator](#)
Successor iterators used by [spot::tgba_explicit](#).
- class [tgba_product](#)
A lazy product. (States are computed on the fly.).
- class [tgba_reachable_iterator](#)
Iterate over all reachable states of a [spot::tgba](#).
- class [tgba_reachable_iterator_breadth_first](#)
An implementation of [spot::tgba_reachable_iterator](#) that browses states breadth first.
- class [tgba_reachable_iterator_depth_first](#)
An implementation of [spot::tgba_reachable_iterator](#) that browses states depth first.
- class [tgba_succ_iterator](#)
Iterate over the successors of a state.
- class [tgba_succ_iterator_concrete](#)
A concrete iterator over successors of a TGBA state.
- class [tgba_succ_iterator_product](#)
Iterate over the successors of a product computed on the fly.
- class [tgba_tba_proxy](#)
Degeneralize a [spot::tgba](#) on the fly.

Typedefs

- typedef std::pair< [yy::Location](#), std::string > [tgba_parse_error](#)
A parse diagnostic with its location.
- typedef std::list< [tgba_parse_error](#) > [tgba_parse_error_list](#)
A list of parser diagnostics, as filled by parse.

Functions

- 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::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 & dotty_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.

- `tgba_explicit * ltl_to_tgba_fm (const ltl::formula *f, bdd_dict *dict)`

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

Save reachable states in text format.

- `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`.

6.1.1 Typedef Documentation

6.1.1.1 typedef std::pair<yy::Location, std::string> spot::tgba_parse_error

A parse diagnostic with its location.

6.1.1.2 typedef std::list<tgba_parse_error> spot::tgba_parse_error_list

A list of parser diagnostics, as filled by parse.

6.1.2 Function Documentation

6.1.2.1 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.

6.1.2.2 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.

6.1.2.3 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.

6.1.2.4 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.

6.1.2.5 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.

6.1.2.6 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.

6.1.2.7 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.

6.1.2.8 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.

6.1.2.9 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.

6.1.2.10 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.

6.1.2.11 const ltl::formula* bdd_to_formula (bdd *f*, const bdd_dict * *d*)**6.1.2.12 std::ostream& dotted_reachable (std::ostream & *os*, const tgba * *g*)**

Print reachable states in dot format.

6.1.2.13 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.

6.1.2.14 bdd formula_to_bdd (const ltl::formula * *f*, bdd_dict * *d*, void * *for_me*)

6.1.2.15 `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 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.

6.1.2.16 `tgba_explicit* ltl_to_tgba_fm (const ltl::formula * f, bdd_dict * dict)`

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}
}
```

6.1.2.17 `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}
}
```


6.1.2.18 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.

6.1.2.19 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.

6.1.2.20 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.

6.1.2.21 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.

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

Save reachable states in text format.

6.1.2.23 const char* version ()

Return Spot's version.

6.2 spot::ltl Namespace Reference**Classes**

- class [atomic_prop](#)

Atomic propositions.

- class `binop`

Binary operator.

- class `clone_visitor`

Clone a formula.

- struct `const_visitor`

Formula visitor that cannot modify the formula.

- class `constant`

A constant (True or False).

- class `default_environment`

A laxist environment.

- class `environment`

An environment that describe atomic propositions.

- class `formula`

An LTL formula.

- class `multop`

Multi-operand operators.

- struct `multop::paircmp`

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

- class `postfix_visitor`

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

- class `ref_formula`

A reference-counted LTL formula.

- class `unabbreviate_logic_visitor`

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

- class `unabbreviate_ltl_visitor`

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

- class `unop`

Unary operator.

- struct `visitor`

Formula visitor that can modify the formula.

Typedefs

- typedef std::pair< [yy::Location](#), std::string > [parse_error](#)
A parse diagnostic with its location.
- typedef std::list< [parse_error](#) > [parse_error_list](#)
A list of parser diagnostics, as filled by parse.

Functions

- [formula](#) * [parse](#) (const std::string <l_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 <l_string, [parse_error_list](#) &error_list)
Format diagnostics produced by [spot::ltl::parse](#).
- [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.
- [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.
- 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.
- [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 Function Documentation

6.2.2.1 formula* clone (const formula *f)

Clone a formula.

6.2.2.2 void destroy (const formula *f)

Destroys a formula.

6.2.2.3 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/>

6.2.2.4 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.

6.2.2.5 bool format_parse_errors (std::ostream &os, const std::string <l_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.

6.2.2.6 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 !*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.)

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

Build a formula from an LTL string.

Parameters:

ltl_string The string to parse.

error_list A list that will be filled with parse errors that occurred during parsing.

env The environment into which parsing should take place.

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

Returns:

A pointer to the formula built from *ltl_string*, or 0 if the input was unparsable.

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

Warning:

This function is not reentrant.

6.2.2.8 std::string to_string (const formula *f)

Convert a formula into a (parsable) string.

Parameters:

f The formula to translate.

6.2.2.9 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.

6.2.2.10 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`.

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

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.

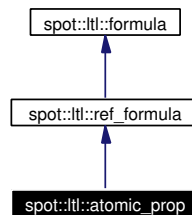
7 spot Class Documentation

7.1 spot::ltl::atomic_prop Class Reference

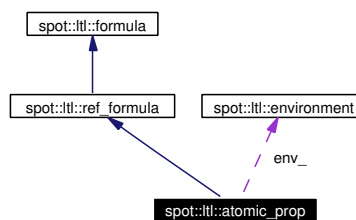
Atomic propositions.

```
#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.
- 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).

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

Get the environment of the atomic proposition.

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

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

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

Number of instantiated atomic propositions. For debugging.

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

Get the name of the atomic proposition.

7.1.4.7 `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.

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

increment reference counter if any

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

7.1.4.9 `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.

7.1.4.10 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:

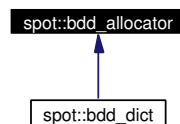
- [atomic_prop.hh](#)

7.2 spot::bdd_allocator Class Reference

Manage ranges of variables.

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

Inheritance diagram for spot::bdd_allocator:

**Protected Types**

- typedef std::pair< int, int > [pos_lenght_pair](#)
- typedef std::list< [pos_lenght_pair](#) > [free_list_type](#)

Protected 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.

Static Protected Member Functions

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

Protected Attributes

- int [lvarnum](#)
number of variables in use in this allocator.
- [free_list_type](#) [free_list](#)
Tracks unused BDD variables.

Static Protected Attributes

- bool [initialized](#)
Whether the BDD library has been initialized.
- int [varnum](#)
number of variables in use in the BDD library.

Private Member Functions

- void [extvarnum](#) (int more)
Require more 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::bdd_allocator::free_list_type` [protected]

7.2.2.2 `typedef std::pair<int, int> spot::bdd_allocator::pos_lenght_pair` [protected]

7.2.3 Constructor & Destructor Documentation

7.2.3.1 `spot::bdd_allocator::bdd_allocator ()` [protected]

Default constructor.

7.2.4 Member Function Documentation

7.2.4.1 int spot::bdd_allocator::allocate_variables (int *n*) [protected]

Allocate *n* BDD variables.

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

Require more variables.

7.2.4.3 void spot::bdd_allocator::initialize () [static, protected]

Initialize the BDD library.

7.2.4.4 void spot::bdd_allocator::release_variables (int *base*, int *n*) [protected]

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

7.2.5 Member Data Documentation

7.2.5.1 free_list_type spot::bdd_allocator::free_list [protected]

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.

7.2.5.4 int spot::bdd_allocator::varnum [static, protected]

number of variables in use in the BDD library.

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

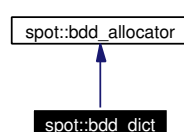
- [bddalloc.hh](#)

7.3 spot::bdd_dict Class Reference

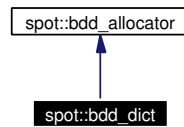
Map BDD variables to formulae.

```
#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.
- 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 the variables used by object.
- std::ostream & [dump](#) (std::ostream &os) const
Dump all variables for debugging.
- void [assert_emptyiness](#) () const
Make sure the dictionary is empty.

- 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)

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 std::pair< int, int > [pos_lenght_pair](#)
- typedef std::list< [pos_lenght_pair](#) > [free_list_type](#)

Protected Member Functions

- int [allocate_variables](#) (int n)
Allocate n BDD variables.
- void [release_variables](#) (int base, int n)
Release n BDD variables starting at base.

Static Protected Member Functions

- void `initialize ()`
Initialize the BDD library.

Protected Attributes

- `vr_map` `var_refs`
- int `lvarnum`
number of variables in use in this allocator.
- `free_list_type` `free_list`
Tracks unused BDD variables.

Static Protected Attributes

- bool `initialized`
Whether the BDD library has been initialized.
- int `varnum`
number of variables in use in the BDD library.

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 std::list<pos_lenght_pair> spot::bdd_allocator::free_list_type` [protected, inherited]

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 std::pair<int, int> spot::bdd_allocator::pos_lenght_pair` [protected, inherited]

7.3.2.4 `typedef Sgi::hash_set<const void*, ptr_hash<void> > spot::bdd_dict::ref_set`
[protected]

BDD-variable reference counts.

7.3.2.5 `typedef Sgi::hash_map<int, const ltl::formula*> spot::bdd_dict::vf_map`

BDD-variable-to-formula maps.

7.3.2.6 `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)` [protected, 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 `void spot::bdd_allocator::initialize ()` [static, protected, inherited]

Initialize the BDD library.

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

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

Check whether formula f has already been registered by by_me .

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

7.3.4.8 `bdd_dict& spot::bdd_dict::operator= (const bdd_dict &other)` [`private`]

7.3.4.9 `int spot::bdd_dict::register_acceptance_variable (const ltl::formula *f, const void *for_me)`

Register an atomic proposition.

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

Returns:

The variable number. Use `bdd_ithvar()` or `bdd_nithvar()` to convert this to a BDD.

7.3.4.10 `void spot::bdd_dict::register_acceptance_variables (bdd f, const void *for_me)`

Register BDD variables as acceptance variables.

Register all variables occurring in *f* as acceptance variables used by *for_me*. This assumes that these acceptance variables are already known from the dictionary (i.e., they have already been registered by `register_acceptance_variable()` for another automaton).

7.3.4.11 `void spot::bdd_dict::register_all_variables_of (const void *from_other, const void *for_me)`

Duplicate the variable usage of another object.

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

7.3.4.12 `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.

7.3.4.13 `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).

7.3.4.14 `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.

7.3.4.15 `void spot::bdd_allocator::release_variables (int base, int n)` [`protected`, `inherited`]

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

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

Release the variables used by object.

Usually called in the destructor if *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_list_type spot::bdd_allocator::free_list` [`protected`, `inherited`]

Tracks unused BDD variables.

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]**7.3.5.13** `int spot::bdd_allocator::varnum` [static, protected, inherited]

number of variables in use in the BDD library.

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

- [bdddict.hh](#)

7.4 `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`

7.4.1 Detailed Description

Comparison functor for BDDs.

7.4.2 Member Function Documentation**7.4.2.1** `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:

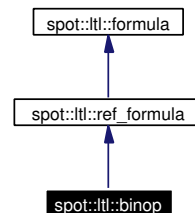
- [bddlt.hh](#)

7.5 spot::ltl::binop Class Reference

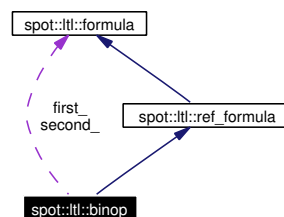
Binary operator.

```
#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).

Static Protected Attributes

- `map instances`

Private Attributes

- `type op_`
- `formula * first_`
- `formula * second_`

7.5.1 Detailed Description

Binary operator.

7.5.2 Member Typedef Documentation

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

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

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

7.5.3 Member Enumeration Documentation**7.5.3.1** `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

7.5.4 Constructor & Destructor Documentation

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

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

7.5.5 Member Function Documentation

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

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

Implements `spot::ltl::formula`.

7.5.5.2 virtual void spot::ltl::binop::accept (visitor & v) [virtual]

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

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

7.5.5.3 formula* spot::ltl::binop::first ()

Get the first operand.

7.5.5.4 const formula* spot::ltl::binop::first () const

Get the first operand.

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

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

7.5.5.6 unsigned spot::ltl::binop::instance_count () [static]

Number of instantiated binary operators. For debugging.

7.5.5.7 type spot::ltl::binop::op () const

Get the type of this operator.

7.5.5.8 const char* spot::ltl::binop::op_name () const

Get the type of this operator, as a string.

7.5.5.9 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.

7.5.5.10 void spot::ltl::ref_formula::ref_ () [protected, virtual, inherited]

increment reference counter if any

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

7.5.5.11 formula* spot::ltl::binop::second ()

Get the second operand.

7.5.5.12 const formula* spot::ltl::binop::second () const

Get the second operand.

7.5.5.13 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.

7.5.5.14 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](#).

7.5.6 Member Data Documentation**7.5.6.1 formula* spot::ltl::binop::first_ [private]****7.5.6.2 map spot::ltl::binop::instances [static, protected]****7.5.6.3 type spot::ltl::binop::op_ [private]****7.5.6.4 formula* spot::ltl::binop::second_ [private]**

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

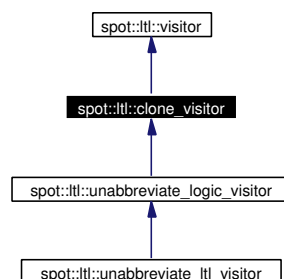
- [binop.hh](#)

7.6 spot::ltl::clone_visitor Class Reference

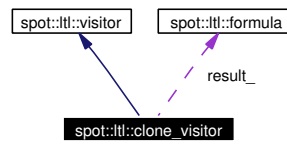
Clone a formula.

```
#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_](#)

7.6.1 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.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 [spot::ltl::clone_visitor::clone_visitor](#) ()

7.6.2.2 virtual [spot::ltl::clone_visitor::~~clone_visitor](#) () [virtual]

7.6.3 Member Function Documentation

7.6.3.1 virtual [formula](#)* [spot::ltl::clone_visitor::recurse](#) ([formula](#) *f) [virtual]

Reimplemented in [spot::ltl::unabbreviate_logic_visitor](#), and [spot::ltl::unabbreviate_ltl_visitor](#).

7.6.3.2 [formula](#)* [spot::ltl::clone_visitor::result](#) () const

7.6.3.3 void [spot::ltl::clone_visitor::visit](#) ([constant](#) *c) [virtual]

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

7.6.3.4 `void spot::ltl::clone_visitor::visit (multop * mo) [virtual]`

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

7.6.3.5 `void spot::ltl::clone_visitor::visit (binop * bo) [virtual]`

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

Reimplemented in [spot::ltl::unabbreviate_logic_visitor](#).

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

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

Reimplemented in [spot::ltl::unabbreviate_ltl_visitor](#).

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

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

7.6.4 Member Data Documentation

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

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

- [clone.hh](#)

7.7 `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

7.7.1 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.

7.7.2 Member Function Documentation

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

7.7.2.2 virtual void spot::ltl::const_visitor::visit (const [unop](#) * node) [pure virtual]

7.7.2.3 virtual void spot::ltl::const_visitor::visit (const [binop](#) * node) [pure virtual]

7.7.2.4 virtual void spot::ltl::const_visitor::visit (const [constant](#) * node) [pure virtual]

7.7.2.5 virtual void spot::ltl::const_visitor::visit (const [atomic_prop](#) * node) [pure virtual]

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

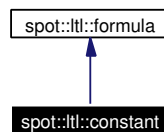
- [visitor.hh](#)

7.8 spot::ltl::constant Class Reference

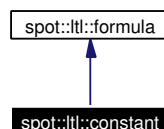
A constant (True or False).

```
#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_`

7.8.1 Detailed Description

A constant (True or False).

7.8.2 Member Enumeration Documentation

7.8.2.1 enum spot::ltl::constant::type

Enumeration values:

False

True

7.8.3 Constructor & Destructor Documentation

7.8.3.1 `spot::ltl::constant::constant (type val)` [protected]

7.8.3.2 `virtual spot::ltl::constant::~~constant ()` [protected, virtual]

7.8.4 Member Function Documentation

7.8.4.1 `virtual void spot::ltl::constant::accept (const_visitor & v) const` [virtual]

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

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

7.8.4.2 `virtual void spot::ltl::constant::accept (visitor & v)` [virtual]

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

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

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

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

7.8.4.4 `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.

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

increment reference counter if any

Reimplemented in [spot::ltl::ref_formula](#).

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

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

7.8.4.7 `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.

7.8.4.8 `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](#).

7.8.4.9 [type spot::ltl::constant::val \(\) const](#)

Return the value of the constant.

7.8.4.10 [const char* spot::ltl::constant::val_name \(\) const](#)

Return the value of the constant as a string.

7.8.5 Member Data Documentation

7.8.5.1 [type spot::ltl::constant::val_](#) [private]

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

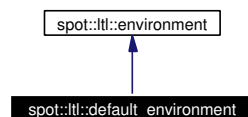
- [constant.hh](#)

7.9 spot::ltl::default_environment Class Reference

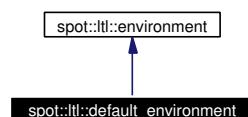
A laxist environment.

```
#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`](#) ()

7.9.1 Detailed Description

A laxist environment.

This environment recognizes all atomic propositions.

This is a singleton. Use [`default_environment::instance\(\)`](#) to obtain the instance.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 `virtual spot::ltl::default_environment::~default_environment ()` [virtual]

7.9.2.2 `spot::ltl::default_environment::default_environment ()` [protected]

7.9.3 Member Function Documentation

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

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

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

Get the name of the environment.

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

7.9.3.3 `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:

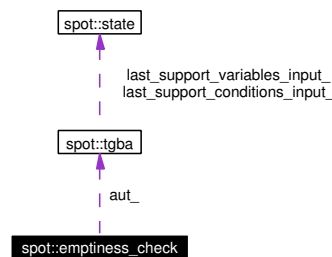
- [defaultenv.hh](#)

7.10 spot::emptiness_check Class Reference

Check whether the language of an automate is empty.

```
#include <emptinesscheck.hh>
```

Collaboration diagram for spot::emptiness_check:



Public Member Functions

- [emptiness_check](#) (const [tgba](#) *a)
- [~emptiness_check](#) ()
- bool [check](#) ()
- void [counter_example](#) ()
Compute a counter example if tgba_emptiness_check() returned false.
- std::ostream & [print_result](#) (std::ostream &os, const [tgba](#) *restrict=0) const

Private Types

- typedef std::list< const [state](#) * > [state_sequence](#)
- typedef std::pair< const [state](#) *, bdd > [state_proposition](#)
- typedef std::list< [state_proposition](#) > [cycle_path](#)
- typedef Sgi::hash_map< const [state](#) *, int, [state_ptr_hash](#), [state_ptr_equal](#) > [hash_type](#)

Private Member Functions

- const [state](#) * [h_filt](#) (const [state](#) *s) const
Return a state which is equal to s, but is in h, and free s if it is different. Doing so simplify memory management, because we don't have to track which state need to be kept or deallocated: all key in h should last for the whole life of the [emptiness_check](#).
- void [remove_component](#) (const [state](#) *start_delete)
Remove a strongly component from the hash.
- void [accepting_path](#) (const [connected_component_set](#) &scc, const [state](#) *start, bdd acc_to_traverse)
- void [complete_cycle](#) (const [connected_component_set](#) &scc, const [state](#) *from, const [state](#) *to)

Private Attributes

- const [tgba](#) * [aut_](#)
- std::stack< [connected_component](#) > [root](#)
- [state_sequence](#) suffix
- [cycle_path](#) period
- [hash_type](#) h

Map of visited states.

7.10.1 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}
}
```

7.10.2 Member Typedef Documentation

7.10.2.1 typedef std::list<[state_proposition](#)> [spot::emptiness_check::cycle_path](#) [private]

7.10.2.2 typedef Sgi::hash_map<const [state*](#), int, [state_ptr_hash](#), [state_ptr_equal](#)> [spot::emptiness_check::hash_type](#) [private]

7.10.2.3 typedef std::pair<const [state*](#), bdd> [spot::emptiness_check::state_proposition](#) [private]

7.10.2.4 typedef std::list<const [state*](#)> [spot::emptiness_check::state_sequence](#) [private]

7.10.3 Constructor & Destructor Documentation

7.10.3.1 [spot::emptiness_check::emptiness_check](#) (const [tgba](#) * *a*)

7.10.3.2 [spot::emptiness_check::~~emptiness_check](#) ()

7.10.4 Member Function Documentation

7.10.4.1 void `spot::emptiness_check::accepting_path` (const `connected_component_set` & *scc*, const `state` * *start*, bdd *acc_to_traverse*) [private]

Called by `counter_example` to find a path which traverses all acceptance conditions in the accepted SCC.

7.10.4.2 bool `spot::emptiness_check::check` ()

This function returns true if the automata's language is empty, and builds a stack of SCC.

7.10.4.3 void `spot::emptiness_check::complete_cycle` (const `connected_component_set` & *scc*, const `state` * *from*, const `state` * *to*) [private]

Complete a cycle that characterise the period of the counter example. Append a sequence to the path given by `accepting_path`.

7.10.4.4 void `spot::emptiness_check::counter_example` ()

Compute a counter example if `tgba_emptiness_check()` returned false.

7.10.4.5 const `state`* `spot::emptiness_check::h_filt` (const `state` * *s*) const [private]

Return a state which is equal to *s*, but is in *h*, and free *s* if it is different. Doing so simplify memory management, because we don't have to track which state need to be kept or deallocated: all key in *h* should last for the whole life of the `emptiness_check`.

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

7.10.4.7 void `spot::emptiness_check::remove_component` (const `state` * *start_delete*) [private]

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.

7.10.5 Member Data Documentation

7.10.5.1 const `tgba`* `spot::emptiness_check::aut_` [private]

7.10.5.2 `hash_type` `spot::emptiness_check::h` [private]

Map of visited states.

7.10.5.3 `cycle_path` `spot::emptiness_check::period` [private]

7.10.5.4 std::stack<`connected_component`> `spot::emptiness_check::root` [private]

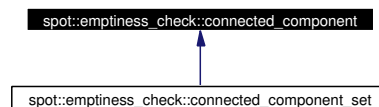
7.10.5.5 state_sequence spot::emptiness_check::suffix [private]

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

- [emptinesscheck.hh](#)

7.11 spot::emptiness_check::connected_component Struct Reference

Inheritance diagram for spot::emptiness_check::connected_component:



Public Member Functions

- [connected_component](#) (int [index](#)=-1)

Public Attributes

- int [index](#)
- bdd [condition](#)

7.11.1 Constructor & Destructor Documentation

7.11.1.1 spot::emptiness_check::connected_component::connected_component (int *index* = -1)

7.11.2 Member Data Documentation

7.11.2.1 bdd spot::emptiness_check::connected_component::condition

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

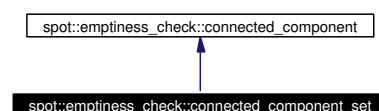
7.11.2.2 int spot::emptiness_check::connected_component::index

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

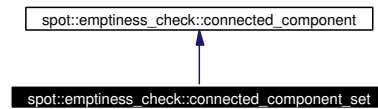
- [emptinesscheck.hh](#)

7.12 spot::emptiness_check::connected_component_set Struct Reference

Inheritance diagram for spot::emptiness_check::connected_component_set:



Collaboration diagram for spot::emptiness_check::connected_component_set:



Public Types

- typedef Sgi::hash_set< const [state](#) *, [state_ptr_hash](#), [state_ptr_equal](#) > [set_type](#)

Public Member Functions

- bool [has_state](#) (const [state](#) *s) const
Check if the SCC contains states s.

Public Attributes

- [set_type](#) [states](#)
- int [index](#)
- bdd [condition](#)

7.12.1 Member Typedef Documentation

7.12.1.1 typedef Sgi::hash_set<const [state](#)*, [state_ptr_hash](#), [state_ptr_equal](#)> [spot::emptiness_check::connected_component_set::set_type](#)

7.12.2 Member Function Documentation

7.12.2.1 bool [spot::emptiness_check::connected_component_set::has_state](#) (const [state](#) * s) const

Check if the SCC contains states *s*.

7.12.3 Member Data Documentation

7.12.3.1 bdd [spot::emptiness_check::connected_component::condition](#) [inherited]

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

7.12.3.2 int [spot::emptiness_check::connected_component::index](#) [inherited]

7.12.3.3 [set_type](#) [spot::emptiness_check::connected_component_set::states](#)

for the counter example we need to know all the states of the component

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

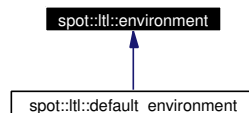
- [emptinesscheck.hh](#)

7.13 spot::ltl::environment Class Reference

An environment that describe 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](#) ()

7.13.1 Detailed Description

An environment that describe atomic propositions.

7.13.2 Constructor & Destructor Documentation

7.13.2.1 virtual spot::ltl::environment::~[~environment](#) () [inline, virtual]

7.13.3 Member Function Documentation

7.13.3.1 virtual const std::string& spot::ltl::environment::name () [pure virtual]

Get the name of the environment.

Implemented in [spot::ltl::default_environment](#).

7.13.3.2 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::default_environment](#).

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

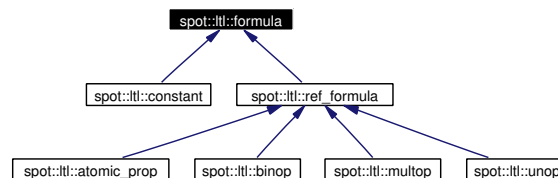
- [environment.hh](#)

7.14 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).

7.14.1 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`.

7.14.2 Constructor & Destructor Documentation

7.14.2.1 virtual `spot::ltl::formula::~~formula()` [virtual]

7.14.3 Member Function Documentation

7.14.3.1 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`.

7.14.3.2 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`.

7.14.3.3 `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.

7.14.3.4 virtual void `spot::ltl::formula::ref_()` [protected, virtual]

increment reference counter if any

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

7.14.3.5 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.

7.14.3.6 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:

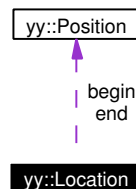
- [formula.hh](#)

7.15 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.

7.15.1 Detailed Description

Abstract a [Location](#).

7.15.2 Constructor & Destructor Documentation

7.15.2.1 yy::Location::Location (void) [inline]

Construct a [Location](#).

7.15.3 Member Function Documentation

7.15.3.1 void yy::Location::columns (unsigned int *count* = 1) [inline]

Extend the current location to the COUNT next columns.

7.15.3.2 void yy::Location::lines (unsigned int *count* = 1) [inline]

Extend the current location to the COUNT next lines.

7.15.3.3 void yy::Location::step (void) [inline]

Reset initial location to final location.

7.15.4 Member Data Documentation

7.15.4.1 Position yy::Location::begin

Beginning of the located region.

7.15.4.2 Position yy::Location::end

End of the located region.

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

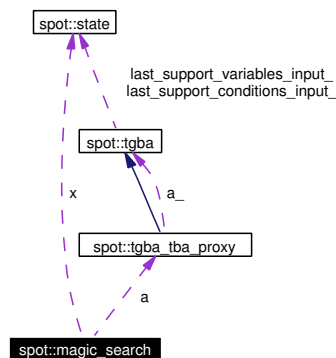
- [location.hh](#)

7.16 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.

7.16.1 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}
}
```

7.16.2 Member Typedef Documentation

7.16.2.1 `typedef Sgi::hash_map<const state*, magic, state_ptr_hash, state_ptr_equal>`
[spot::magic_search::hash_type](#) [private]

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

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

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

7.16.3 Constructor & Destructor Documentation

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

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

7.16.3.2 `spot::magic_search::~~magic_search ()`

7.16.4 Member Function Documentation

7.16.4.1 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.

7.16.4.2 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*.

7.16.4.3 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.

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

Append a new state to the current path.

7.16.5 Member Data Documentation

7.16.5.1 const [tgba_tba_proxy](#)* spot::magic_search::a [private]

The automata to check.

7.16.5.2 [hash_type](#) spot::magic_search::h [private]

Map of visited states.

7.16.5.3 [stack_type](#) spot::magic_search::stack [private]

Stack of visited states on the path.

7.16.5.4 [tstack_type](#) spot::magic_search::tstack [private]

Stack of transitions.

This is an addition to the data from the paper.

7.16.5.5 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:

- [magic.hh](#)

7.17 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

7.17.1 Detailed Description

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

7.17.2 Member Data Documentation

7.17.2.1 bool [spot::magic_search::magic::seen_with](#)

7.17.2.2 bool [spot::magic_search::magic::seen_without](#)

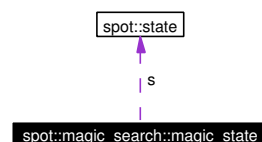
The documentation for this struct was generated from the following file:

- [magic.hh](#)

7.18 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.

7.18.1 Detailed Description

A state for the [spot::magic_search](#) algorithm.

7.18.2 Member Data Documentation

7.18.2.1 bool [spot::magic_search::magic_state::m](#)

The state of the magic demon.

7.18.2.2 const state* [spot::magic_search::magic_state::s](#)

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

- [magic.hh](#)

7.19 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_](#)

7.19.1 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}
}

```

7.19.2 Constructor & Destructor Documentation

7.19.2.1 `spot::minato_isop::minato_isop (bdd input)`

Constructor.

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

7.19.2.2 `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*.

7.19.3 Member Function Documentation

7.19.3.1 `bdd spot::minato_isop::next ()`

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

7.19.4 Member Data Documentation

7.19.4.1 `std::stack<bdd> spot::minato_isop::cube_ [private]`

7.19.4.2 `bdd spot::minato_isop::ret_ [private]`

7.19.4.3 `std::stack<local_vars> spot::minato_isop::todo_ [private]`

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

- [minato.hh](#)

7.20 `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](#)

7.20.1 Detailed Description

Internal variables for [minato_isop](#).

7.20.2 Member Enumeration Documentation

7.20.2.1 anonymous enum

Enumeration values:

FirstStep

SecondStep

ThirdStep

FourthStep

7.20.3 Constructor & Destructor Documentation

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

7.20.4 Member Data Documentation

7.20.4.1 bdd [spot::minato_isop::local_vars::f0_max](#)

7.20.4.2 bdd [spot::minato_isop::local_vars::f0_min](#)

7.20.4.3 bdd [spot::minato_isop::local_vars::f1_max](#)

7.20.4.4 bdd [spot::minato_isop::local_vars::f1_min](#)

7.20.4.5 bdd [spot::minato_isop::local_vars::f_max](#)

7.20.4.6 bdd [spot::minato_isop::local_vars::f_min](#)

7.20.4.7 bdd [spot::minato_isop::local_vars::g0](#)

7.20.4.8 bdd [spot::minato_isop::local_vars::g1](#)

7.20.4.9 enum { ... } [spot::minato_isop::local_vars::step](#)

7.20.4.10 bdd [spot::minato_isop::local_vars::v1](#)

7.20.4.11 bdd [spot::minato_isop::local_vars::vars](#)

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

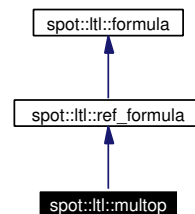
- [minato.hh](#)

7.21 spot::ltl::multop Class Reference

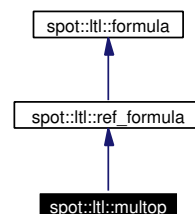
Multi-operand operators.

```
#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 nth children.
- formula * nth (unsigned n)
Get the nth children.
- type op () const
Get the type of this operator.
- const char * op_name () const
Get the type of this operator, as a string.
- formula * ref ()
clone this node

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](#) *, [pairecmp](#) > [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).

Static Protected Attributes

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

Private Attributes

- [type](#) [op_](#)
- [vec](#) * [children_](#)

7.21.1 Detailed Description

Multi-operand operators.

These operators are considered commutative and associative.

7.21.2 Member Typedef Documentation

7.21.2.1 typedef std::map<[pair](#), [formula](#)*, [pairecmp](#)> [spot::ltl::multop::map](#) [protected]

7.21.2.2 typedef std::pair<[type](#), [vec](#)*> [spot::ltl::multop::pair](#) [protected]

7.21.2.3 typedef std::vector<[formula](#)*> [spot::ltl::multop::vec](#)

List of formulae.

7.21.3 Member Enumeration Documentation

7.21.3.1 enum [spot::ltl::multop::type](#)

Enumeration values:

Or

And

7.21.4 Constructor & Destructor Documentation

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

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

7.21.5 Member Function Documentation

7.21.5.1 `virtual void spot::ltl::multop::accept (const_visitor & v) const` [virtual]

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

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

7.21.5.2 `virtual void spot::ltl::multop::accept (visitor & v)` [virtual]

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

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

7.21.5.3 `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.

7.21.5.4 `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.

7.21.5.5 `unsigned spot::ltl::multop::instance_count ()` [static]

Number of instantiated multi-operand operators. For debugging.

7.21.5.6 `formula* spot::ltl::multop::nth (unsigned n)`

Get the *n*th children.

Starting with *n* = 0.

7.21.5.7 `const formula* spot::ltl::multop::nth (unsigned n) const`

Get the *n*th children.

Starting with $n = 0$.

7.21.5.8 `type spot::ltl::multop::op () const`

Get the type of this operator.

7.21.5.9 `const char* spot::ltl::multop::op_name () const`

Get the type of this operator, as a string.

7.21.5.10 `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.

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

increment reference counter if any

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

7.21.5.12 `unsigned spot::ltl::multop::size () const`

Get the number of children.

7.21.5.13 `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.

7.21.5.14 `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`.

7.21.6 Member Data Documentation**7.21.6.1** `vec* spot::ltl::multop::children_ [private]`**7.21.6.2** `map spot::ltl::multop::instances [static, protected]`

7.21.6.3 type spot::ltl::multop::op_ [private]

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

- [multop.hh](#)

7.22 spot::ltl::multop::paircmp Struct Reference

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

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

Public Member Functions

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

7.22.1 Detailed Description

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

7.22.2 Member Function Documentation

7.22.2.1 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:

- [multop.hh](#)

7.23 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.

7.23.1 Detailed Description

Abstract a [Position](#).

7.23.2 Constructor & Destructor Documentation

7.23.2.1 yy::Position::Position () [inline]

Construct a [Position](#).

7.23.3 Member Function Documentation

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

(column related) Advance to the COUNT next columns.

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

(line related) Advance to the COUNT next lines.

7.23.4 Member Data Documentation

7.23.4.1 unsigned int yy::Position::column

Current column number.

7.23.4.2 std::string yy::Position::filename

File name to which this position refers.

7.23.4.3 `const unsigned int yy::Position::initial_column = 0` [static]

Initial column number.

7.23.4.4 `const unsigned int yy::Position::initial_line = 1` [static]

Initial line number.

7.23.4.5 `unsigned int yy::Position::line`

Current line number.

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

- [position.hh](#)

7.24 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)`

7.24.1 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.

7.24.2 Constructor & Destructor Documentation

7.24.2.1 spot::ltl::postfix_visitor::postfix_visitor ()

7.24.2.2 virtual spot::ltl::postfix_visitor::~~postfix_visitor () [virtual]

7.24.3 Member Function Documentation

7.24.3.1 virtual void spot::ltl::postfix_visitor::doit (constant * c) [virtual]

7.24.3.2 virtual void spot::ltl::postfix_visitor::doit (multop * mo) [virtual]

7.24.3.3 virtual void spot::ltl::postfix_visitor::doit (binop * bo) [virtual]

7.24.3.4 virtual void spot::ltl::postfix_visitor::doit (unop * uo) [virtual]

7.24.3.5 virtual void spot::ltl::postfix_visitor::doit (atomic_prop * ap) [virtual]

7.24.3.6 virtual void spot::ltl::postfix_visitor::doit_default (formula * f) [virtual]

7.24.3.7 void spot::ltl::postfix_visitor::visit (constant * c) [virtual]

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

7.24.3.8 void spot::ltl::postfix_visitor::visit (multop * mo) [virtual]

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

7.24.3.9 void spot::ltl::postfix_visitor::visit (binop * bo) [virtual]

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

7.24.3.10 void spot::ltl::postfix_visitor::visit (unop * uo) [virtual]

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

7.24.3.11 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:

- [postfix.hh](#)

7.25 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`

7.25.1 Detailed Description

```
template<class T> struct spot::ptr_hash< T >
```

A hash function for pointers.

7.25.2 Member Function Documentation

7.25.2.1 `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:

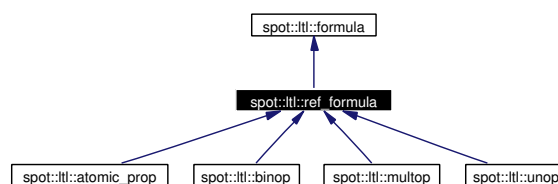
- [hash.hh](#)

7.26 spot::ltl::ref_formula Class Reference

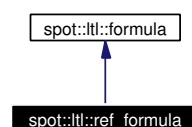
A reference-counted LTL formula.

```
#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).

Private Attributes

- unsigned `ref_count_`

7.26.1 Detailed Description

A reference-counted LTL formula.

7.26.2 Constructor & Destructor Documentation

7.26.2.1 virtual `spot::ltl::ref_formula::~~ref_formula ()` [protected, virtual]

7.26.2.2 `spot::ltl::ref_formula::ref_formula ()` [protected]

7.26.3 Member Function Documentation

7.26.3.1 `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`.

7.26.3.2 `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`.

7.26.3.3 `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.

7.26.3.4 `void spot::ltl::ref_formula::ref_ ()` [protected, virtual]

increment reference counter if any

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

7.26.3.5 `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.

7.26.3.6 `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`.

7.26.4 Member Data Documentation

7.26.4.1 `unsigned spot::ltl::ref_formula::ref_count_` [private]

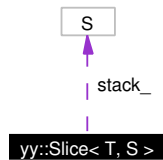
The documentation for this class was generated from the following file:

- `reformula.hh`

7.27 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 >
```

7.27.1 Constructor & Destructor Documentation

7.27.1.1 `template<class T, class S = Stack< T >> yy::Slice< T, S >::Slice (const S & stack, unsigned range)` `[inline]`

7.27.2 Member Function Documentation

7.27.2.1 `[]` `template<class T, class S = Stack< T >> const T& yy::Slice< T, S >::operator\[\] (unsigned i) const` `[inline]`

7.27.3 Member Data Documentation

7.27.3.1 `template<class T, class S = Stack< T >> unsigned yy::Slice< T, S >::range_` `[private]`

7.27.3.2 `template<class T, class S = Stack< T >> const S& yy::Slice< T, S >::stack_` `[private]`

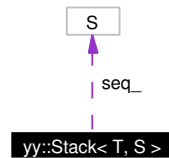
The documentation for this class was generated from the following file:

- [stack.hh](#)

7.28 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 >
```

7.28.1 Member Typedef Documentation

7.28.1.1 template<class T, class S = std::deque< T >> typedef S::const_iterator [yy::Stack](#)< T, S >::[ConstIterator](#)

7.28.1.2 template<class T, class S = std::deque< T >> typedef S::iterator [yy::Stack](#)< T, S >::[Iterator](#)

7.28.2 Constructor & Destructor Documentation

7.28.2.1 template<class T, class S = std::deque< T >> [yy::Stack](#)< T, S >::[Stack](#) () [inline]

7.28.2.2 `template<class T, class S = std::deque< T >> yy::Stack< T, S >::Stack (unsigned n)`
`[inline]`

7.28.3 Member Function Documentation

7.28.3.1 `template<class T, class S = std::deque< T >> ConstIterator yy::Stack< T, S >::begin ()`
`const [inline]`

7.28.3.2 `template<class T, class S = std::deque< T >> ConstIterator yy::Stack< T, S >::end ()`
`const [inline]`

7.28.3.3 `template<class T, class S = std::deque< T >> unsigned yy::Stack< T, S >::height () const`
`[inline]`

7.28.3.4 `] template<class T, class S = std::deque< T >> const T& yy::Stack< T, S >::operator[]`
`(unsigned i) const [inline]`

7.28.3.5 `] template<class T, class S = std::deque< T >> T& yy::Stack< T, S >::operator[] (unsigned`
`i) [inline]`

7.28.3.6 `template<class T, class S = std::deque< T >> void yy::Stack< T, S >::pop (unsigned n =`
`1) [inline]`

7.28.3.7 `template<class T, class S = std::deque< T >> void yy::Stack< T, S >::push (const T & t)`
`[inline]`

7.28.4 Member Data Documentation

7.28.4.1 `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:

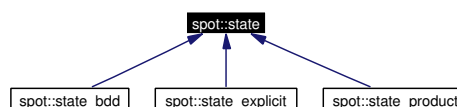
- [stack.hh](#)

7.29 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` ()

7.29.1 Detailed Description

Abstract class for states.

7.29.2 Constructor & Destructor Documentation

7.29.2.1 virtual `spot::state::~~state` () [inline, virtual]

7.29.3 Member Function Documentation

7.29.3.1 virtual `state`* `spot::state::clone` () const [pure virtual]

Duplicate a state.

Implemented in `spot::state_bdd`, `spot::state_explicit`, and `spot::state_product`.

7.29.3.2 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`.

7.29.3.3 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 as long as 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:

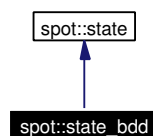
- [state.hh](#)

7.30 spot::state_bdd Class Reference

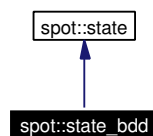
A state whose representation is a BDD.

```
#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.

7.30.1 Detailed Description

A state whose representation is a BDD.

7.30.2 Constructor & Destructor Documentation

7.30.2.1 `spot::state_bdd::state_bdd (bdd s)` [inline]

7.30.3 Member Function Documentation

7.30.3.1 `virtual bdd spot::state_bdd::as_bdd () const` [inline, virtual]

Return the BDD part of the state.

7.30.3.2 `virtual state_bdd* spot::state_bdd::clone () const` [virtual]

Duplicate a state.

Implements `spot::state`.

7.30.3.3 `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`.

7.30.3.4 `virtual size_t spot::state_bdd::hash () const` [virtual]

Hash a state.

This method returns an integer that can be used as a hash value for this state.

Note that the hash value is guaranteed to be unique for all equal states (in `compare()`'s sense) for only has long has one of these states exists. So it's OK to use a `spot::state` as a key in a `hash_map` because the mere use of the state as a key in the hash will ensure the state continues to exist.

However if you create the state, get its hash key, delete the state, recreate the same state, and get its hash key, you may obtain two different hash keys if the same state were not already used elsewhere. In practice this weird situation can occur only when the state is BDD-encoded, because BDD numbers (used to build the hash value) can be reused for other formulas. That probably doesn't matter, since the hash value is meant to be used in a `hash_map`, but it had to be noted.

Implements [spot::state](#).

7.30.4 Member Data Documentation

7.30.4.1 bdd [spot::state_bdd::state_](#) [protected]

BDD representation of the state.

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

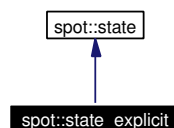
- [statebdd.hh](#)

7.31 spot::state_explicit Class Reference

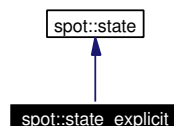
States used by [spot::tgba_explicit](#).

```
#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_`

7.31.1 Detailed Description

States used by `spot::tgba_explicit`.

7.31.2 Constructor & Destructor Documentation

7.31.2.1 `spot::state_explicit::state_explicit (const tgba_explicit::state * s)` `[inline]`

7.31.2.2 `virtual spot::state_explicit::~~state_explicit ()` `[inline, virtual]`

7.31.3 Member Function Documentation

7.31.3.1 `virtual state_explicit* spot::state_explicit::clone () const` `[virtual]`

Duplicate a state.

Implements `spot::state`.

7.31.3.2 `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`.

7.31.3.3 `virtual int spot::state_explicit::compare (const spot::state * other) const` `[virtual]`

7.31.3.4 `const tgba_explicit::state* spot::state_explicit::get_state () const`

7.31.3.5 `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](#).

7.31.4 Member Data Documentation

7.31.4.1 `const tgba_explicit::state* spot::state_explicit::state_ [private]`

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

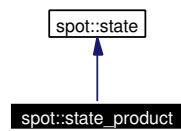
- [tgbaexplicit.hh](#)

7.32 spot::state_product Class Reference

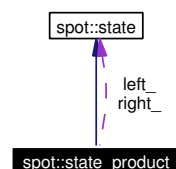
A state for [spot::tgba_product](#).

```
#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.

7.32.1 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.

7.32.2 Constructor & Destructor Documentation

7.32.2.1 `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.

7.32.2.2 `spot::state_product::state_product (const state_product & o)`

Copy constructor.

7.32.2.3 `virtual spot::state_product::~~state_product () [virtual]`

7.32.3 Member Function Documentation

7.32.3.1 `virtual state_product* spot::state_product::clone () const [virtual]`

Duplicate a state.

Implements `spot::state`.

7.32.3.2 `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](#).

7.32.3.3 `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](#).

7.32.3.4 `state* spot::state_product::left () const` `[inline]`**7.32.3.5** `state* spot::state_product::right () const` `[inline]`**7.32.4 Member Data Documentation****7.32.4.1** `state* spot::state_product::left_` `[private]`

State from the left automaton.

7.32.4.2 `state* spot::state_product::right_` `[private]`

State from the right automaton.

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

- [tgbaproduct.hh](#)

7.33 `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

7.33.1 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;
```

7.33.2 Member Function Documentation

7.33.2.1 `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:

- [state.hh](#)

7.34 spot::state_ptr_hash Struct Reference

Hash Function for `state*`.

```
#include <state.hh>
```

Public Member Functions

- `size_t operator()` (const `state` *that) const

7.34.1 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;
```

7.34.2 Member Function Documentation

7.34.2.1 `size_t spot::state_ptr_hash::operator()` (const `state` * *that*) const [inline]

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

- [state.hh](#)

7.35 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

7.35.1 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;
```

7.35.2 Member Function Documentation

7.35.2.1 `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:

- [state.hh](#)

7.36 spot::string_hash Struct Reference

A hash function for strings.

```
#include <hash.hh>
```

Public Member Functions

- `size_t operator()` (const std::string &s) const

7.36.1 Detailed Description

A hash function for strings.

7.36.2 Member Function Documentation

7.36.2.1 `size_t spot::string_hash::operator()` (const std::string & *s*) const [inline]

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

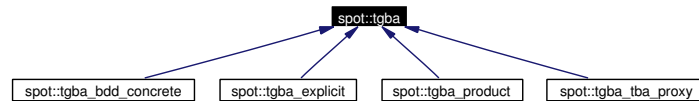
- [hash.hh](#)

7.37 spot::tgba Class Reference

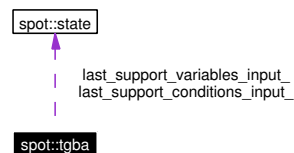
A Transition-based Generalized Büchi Automaton.

```
#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_`

7.37.1 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.

7.37.2 Constructor & Destructor Documentation

7.37.2.1 `spot::tgba::tgba ()` [protected]

7.37.2.2 `virtual spot::tgba::~~tgba ()` [virtual]

7.37.3 Member Function Documentation

7.37.3.1 `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](#).

7.37.3.2 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](#).

7.37.3.3 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](#).

7.37.3.4 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](#).

7.37.3.5 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](#).

7.37.3.6 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](#).

7.37.3.7 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`.

7.37.3.8 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`.

7.37.3.9 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 state of the global product automaton. Otherwise, 0.

Implemented in `spot::tgba_bdd_concrete`, `spot::tgba_product`, and `spot::tgba_tba_proxy`.

7.37.3.10 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.

7.37.3.11 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.

7.37.4 Member Data Documentation

7.37.4.1 const state* spot::tgba::last_support_conditions_input_ [mutable, private]

7.37.4.2 bdd spot::tgba::last_support_conditions_output_ [mutable, private]

7.37.4.3 const state* spot::tgba::last_support_variables_input_ [mutable, private]

7.37.4.4 bdd spot::tgba::last_support_variables_output_ [mutable, private]

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

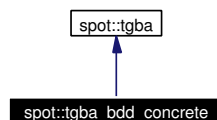
- [tgba.hh](#)

7.38 spot::tgba_bdd_concrete Class Reference

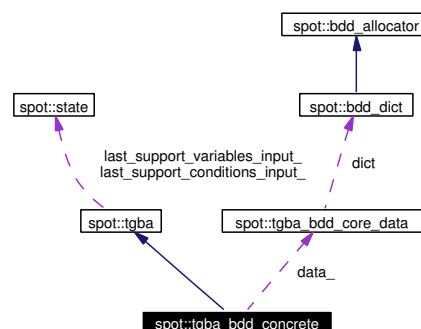
A concrete `spot::tgba` implemented using BDDs.

```
#include <tgbabddconcrete.hh>
```

Inheritance diagram for `spot::tgba_bdd_concrete`:



Collaboration diagram for `spot::tgba_bdd_concrete`:



Public Member Functions

- [tgba_bdd_concrete](#) (const [tgba_bdd_factory](#) &fact)
Construct a [tgba_bdd_concrete](#) with unknown initial state.
- [tgba_bdd_concrete](#) (const [tgba_bdd_factory](#) &fact, bdd init)
Construct a [tgba_bdd_concrete](#) with known initial state.
- virtual [~tgba_bdd_concrete](#) ()
- virtual void [set_init_state](#) (bdd s)
Set the initial state.
- virtual [state_bdd](#) * [get_init_state](#) () const
Get the initial state of the automaton.
- bdd [get_init_bdd](#) () const
Get the initial state directly as a BDD.
- virtual [tgba_succ_iterator_concrete](#) * [succ_iter](#) (const [state](#) *local_state, const [state](#) *global_state=0, const [tgba](#) *global_automaton=0) const
Get an iterator over the successors of local_state.
- virtual std::string [format_state](#) (const [state](#) *state) const
Format the state as a string for printing.
- virtual [bdd_dict](#) * [get_dict](#) () const
Get the dictionary associated to the automaton.
- const [tgba_bdd_core_data](#) & [get_core_data](#) () const
Get the core data associated to this automaton.
- virtual bdd [all_acceptance_conditions](#) () const
Return the set of all acceptance conditions used by this automaton.
- virtual bdd [neg_acceptance_conditions](#) () const
Return the conjunction of all negated acceptance variables.
- bdd [support_conditions](#) (const [state](#) *state) const
Get a formula that must hold whatever successor is taken.
- bdd [support_variables](#) (const [state](#) *state) const
Get the conjunctions of variables tested by the outgoing transitions of state.
- virtual [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](#) &)

7.38.1 Detailed Description

A concrete [spot::tgba](#) implemented using BDDs.

7.38.2 Constructor & Destructor Documentation

7.38.2.1 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.

7.38.2.2 spot::tgba_bdd_concrete::tgba_bdd_concrete (const [tgba_bdd_factory](#) & fact, bdd init)

Construct a [tgba_bdd_concrete](#) with known initial state.

7.38.2.3 virtual spot::tgba_bdd_concrete::~~[tgba_bdd_concrete](#) () [virtual]

7.38.2.4 spot::tgba_bdd_concrete::tgba_bdd_concrete (const [tgba_bdd_concrete](#) &) [private]

7.38.3 Member Function Documentation

7.38.3.1 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 accepting conditions. I.e., the union of the accepting conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::tgba](#).

7.38.3.2 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](#).

7.38.3.3 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](#).

7.38.3.4 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](#).

7.38.3.5 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.

7.38.3.6 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](#).

7.38.3.7 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;
```

7.38.3.8 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](#).

7.38.3.9 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](#).

7.38.3.10 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](#).

7.38.3.11 virtual void spot::tgba_bdd_concrete::set_init_state (bdd s) [virtual]

Set the initial state.

7.38.3.12 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 state of the global product automaton. Otherwise, 0.

Implements [spot::tgba](#).

7.38.3.13 `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.

7.38.3.14 `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.

7.38.3.15 `tgba_bdd_concrete& spot::tgba_bdd_concrete::tgba_bdd_concrete::operator= (const tgba_bdd_concrete &) [private]`

7.38.4 Member Data Documentation

7.38.4.1 `tgba_bdd_core_data spot::tgba_bdd_concrete::data_` [protected]

Core data associated to the automaton.

7.38.4.2 `bdd spot::tgba_bdd_concrete::init_` [protected]

Initial state.

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

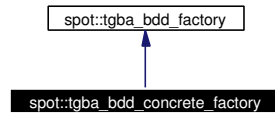
- [tgbabddconcrete.hh](#)

7.39 `spot::tgba_bdd_concrete_factory` Class Reference

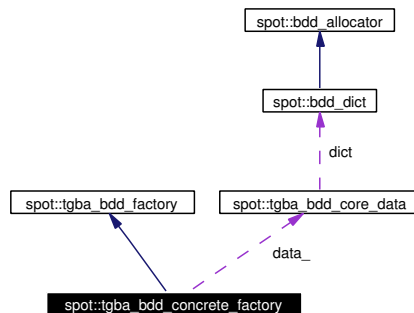
Helper class to build a [spot::tgba_bdd_concrete](#) object.

```
#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.

7.39.1 Detailed Description

Helper class to build a [spot::tgba_bdd_concrete](#) object.

7.39.2 Member Typedef Documentation

7.39.2.1 `typedef Sgi::hash_map<const ltl::formula*, bdd, ptr_hash<ltl::formula> > spot::tgba_bdd_concrete_factory::acc_map_ [private]`

7.39.3 Constructor & Destructor Documentation

7.39.3.1 `spot::tgba_bdd_concrete_factory::tgba_bdd_concrete_factory (bdd_dict * dict)`

7.39.3.2 `virtual spot::tgba_bdd_concrete_factory::~~tgba_bdd_concrete_factory () [virtual]`

7.39.4 Member Function Documentation

7.39.4.1 `void spot::tgba_bdd_concrete_factory::constrain_relation (bdd new_rel)`

Add a new constraint to the relation.

7.39.4.2 `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()`.

7.39.4.3 `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()`.

7.39.4.4 `void spot::tgba_bdd_concrete_factory::declare_acceptance_condition (bdd b, const ltl::formula * a)`

Declare an acceptance condition.

Formula such as 'f 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

7.39.4.5 `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\(\)](#).

7.39.4.6 `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](#).

7.39.4.7 `bdd_dict* spot::tgba_bdd_concrete_factory::get_dict () const`

7.39.5 Member Data Documentation

7.39.5.1 `acc_map spot::tgba_bdd_concrete_factory::acc_` [private]

BDD associated to each acceptance condition.

7.39.5.2 `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:

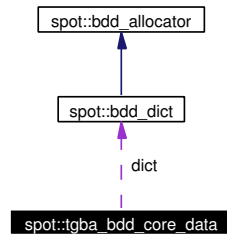
- [tgbabddconcretefactory.hh](#)

7.40 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 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](#) ©)
- void [declare_now_next](#) (bdd now, bdd next)
Update the variable sets to take a new pair of variables into account.
- void [declare_atomic_prop](#) (bdd var)
Update the variable sets to take a new atomic proposition into account.
- void [declare_acceptance_condition](#) (bdd prom)
Update the variable sets to take a new acceptance condition into account.

Public Attributes

- bdd [relation](#)
encodes the transition relation of the TGBA.
- bdd [acceptance_conditions](#)
encodes the acceptance conditions
- bdd [all_acceptance_conditions](#)
The set of all acceptance conditions used by the Automaton.
- bdd [now_set](#)
The conjunction of all Now variables, in their positive form.
- bdd [next_set](#)
The conjunction of all Next variables, in their positive form.
- bdd [nownext_set](#)

The conjunction of all Now and Next variables, in their positive form.

- bdd [notnow_set](#)

The (positive) conjunction of all variables which are not Now variables.

- bdd [notnext_set](#)

The (positive) conjunction of all variables which are not Next variables.

- bdd [var_set](#)

The (positive) conjunction of all variables which are atomic propositions.

- bdd [notvar_set](#)

The (positive) conjunction of all variables which are not atomic propositions.

- bdd [varandnext_set](#)

The (positive) conjunction of all Next variables and atomic propositions.

- bdd [acc_set](#)

The (positive) conjunction of all variables which are acceptance conditions.

- bdd [notacc_set](#)

The (positive) conjunction of all variables which are not acceptance conditions.

- bdd [negacc_set](#)

The negative conjunction of all variables which are acceptance conditions.

- [bdd_dict](#) * [dict](#)

The dictionary used by the automata.

7.40.1 Detailed Description

Core data for a TGBA encoded using BDDs.

7.40.2 Constructor & Destructor Documentation

7.40.2.1 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.

7.40.2.2 spot::tgba_bdd_core_data::tgba_bdd_core_data (const [tgba_bdd_core_data](#) & *copy*)

Copy constructor.

7.40.2.3 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.

7.40.3 Member Function Documentation

7.40.3.1 void spot::tgba_bdd_core_data::declare_acceptance_condition (bdd *prom*)

Update the variable sets to take a new acceptance condition into account.

7.40.3.2 void spot::tgba_bdd_core_data::declare_atomic_prop (bdd *var*)

Update the variable sets to take a new atomic proposition into account.

7.40.3.3 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.

7.40.3.4 const tgba_bdd_core_data& spot::tgba_bdd_core_data::operator= (const tgba_bdd_core_data & *copy*)

7.40.4 Member Data Documentation

7.40.4.1 bdd spot::tgba_bdd_core_data::acc_set

The (positive) conjunction of all variables which are acceptance conditions.

7.40.4.2 bdd spot::tgba_bdd_core_data::acceptance_conditions

encodes the acceptance conditions

$a \cup b$, or $F b$, 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 $F b$.)

The `spot::succ_iter::current_acceptance_conditions()` method will return the $Acc[x]$ variables of the acceptance sets in which a transition is. Actually we never return $Acc[x]$ alone, but $Acc[x]$ and all other acceptance variables negated.

So if there is three acceptance set a , b , and c , and a transition is in set a , we'll return $Acc[a] \& !Acc[b] \& !Acc[c]$. If the transition is in both a and b , we'll return $(Acc[a] \& !Acc[b] \& !Acc[c]) \mid (!Acc[a] \& Acc[b] \& !Acc[c])$.

Accepting conditions are attributed to transitions and are only concerned by atomic propositions (which label the transitions) and Next variables (the destination). Typically, a transition should bear the variable $Acc[b]$ if it doesn't check for 'b' and have a destination of the form $a \cup b$, or $F b$.

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.

7.40.4.3 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.

7.40.4.4 bdd_dict* spot::tgba_bdd_core_data::dict

The dictionary used by the automata.

7.40.4.5 bdd spot::tgba_bdd_core_data::negacc_set

The negative conjunction of all variables which are acceptance conditions.

7.40.4.6 bdd spot::tgba_bdd_core_data::next_set

The conjunction of all Next variables, in their positive form.

7.40.4.7 bdd spot::tgba_bdd_core_data::notacc_set

The (positive) conjunction of all variables which are not acceptance conditions.

7.40.4.8 bdd spot::tgba_bdd_core_data::notnext_set

The (positive) conjunction of all variables which are not Next variables.

7.40.4.9 bdd spot::tgba_bdd_core_data::notnow_set

The (positive) conjunction of all variables which are not Now variables.

7.40.4.10 bdd spot::tgba_bdd_core_data::notvar_set

The (positive) conjunction of all variables which are not atomic propositions.

7.40.4.11 bdd spot::tgba_bdd_core_data::now_set

The conjunction of all Now variables, in their positive form.

7.40.4.12 bdd spot::tgba_bdd_core_data::nownext_set

The conjunction of all Now and Next variables, in their positive form.

7.40.4.13 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

7.40.4.14 bdd [spot::tgba_bdd_core_data::var_set](#)

The (positive) conjunction of all variables which are atomic propositions.

7.40.4.15 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:

- [tgbabddcoredata.hh](#)

7.41 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.

7.41.1 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.

7.41.2 Member Function Documentation

7.41.2.1 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:

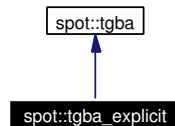
- [tgbabddfactory.hh](#)

7.42 spot::tgba_explicit Class Reference

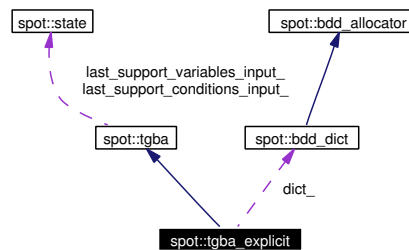
Explicit representation of a [spot::tgba](#).

```
#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](#) ()
- 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*)

7.42.1 Detailed Description

Explicit representation of a [spot::tgba](#).

7.42.2 Member Typedef Documentation

7.42.2.1 `typedef Sgi::hash_map<const std::string, tgba_explicit::state*, string_hash> spot::tgba_explicit::ns_map [protected]`

7.42.2.2 `typedef Sgi::hash_map<const tgba_explicit::state*, std::string, ptr_hash<tgba_explicit::state>> spot::tgba_explicit::sn_map [protected]`

7.42.2.3 `typedef std::list<transition*> spot::tgba_explicit::state`

7.42.3 Constructor & Destructor Documentation

7.42.3.1 `spot::tgba_explicit::tgba_explicit (bdd_dict * dict)`

7.42.3.2 `virtual spot::tgba_explicit::~~tgba_explicit () [virtual]`

7.42.3.3 `spot::tgba_explicit::tgba_explicit (const tgba_explicit & other) [private]`

7.42.4 Member Function Documentation

7.42.4.1 `void spot::tgba_explicit::add_acceptance_condition (transition * t, const ltl::formula * f)`

7.42.4.2 `void spot::tgba_explicit::add_acceptance_conditions (transition * t, bdd f)`

This assumes that all acceptance conditions in *f* are known from *dict*.

7.42.4.3 void spot::tgba_explicit::add_condition (transition * *t*, const ltl::formula * *f*)

7.42.4.4 void spot::tgba_explicit::add_conditions (transition * *t*, bdd *f*)

This assumes that all variables in *f* are known from dict.

7.42.4.5 state* spot::tgba_explicit::add_state (const std::string & *name*) [protected]

7.42.4.6 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](#).

7.42.4.7 void spot::tgba_explicit::complement_all_acceptance_conditions ()

7.42.4.8 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](#).

7.42.4.9 virtual bdd spot::tgba_explicit::compute_support_conditions (const spot::state * *state*) const [protected, virtual]

7.42.4.10 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](#).

7.42.4.11 virtual bdd spot::tgba_explicit::compute_support_variables (const spot::state * *state*) const [protected, virtual]

7.42.4.12 transition* spot::tgba_explicit::create_transition (const std::string & *source*, const std::string & *dest*)

7.42.4.13 void spot::tgba_explicit::declare_acceptance_condition (const ltl::formula * *f*)

7.42.4.14 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](#).

7.42.4.15 `virtual std::string spot::tgba_explicit::format_state (const spot::state * state) const` [virtual]

7.42.4.16 `bdd spot::tgba_explicit::get_acceptance_condition (const ltl::formula * f)` [protected]

7.42.4.17 `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](#).

7.42.4.18 `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](#).

7.42.4.19 `bool spot::tgba_explicit::has_acceptance_condition (const ltl::formula * f) const`

7.42.4.20 `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](#).

7.42.4.21 `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](#).

7.42.4.22 void spot::tgba_explicit::set_init_state (const std::string & state)

7.42.4.23 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 state of the global product automaton. Otherwise, 0.

Implemented in [spot::tgba_bdd_concrete](#), [spot::tgba_product](#), and [spot::tgba_tba_proxy](#).

7.42.4.24 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]

7.42.4.25 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.

7.42.4.26 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.

7.42.4.27 tgba_explicit& spot::tgba_explicit::tgba_explicit::operator= (const tgba_explicit & other) [private]

7.42.5 Member Data Documentation

7.42.5.1 `bdd spot::tgba_explicit::all_acceptance_conditions_` [mutable, protected]

7.42.5.2 `bool spot::tgba_explicit::all_acceptance_conditions_computed_` [mutable, protected]

7.42.5.3 `bdd_dict* spot::tgba_explicit::dict_` [protected]

7.42.5.4 `tgba_explicit::state* spot::tgba_explicit::init_` [protected]

7.42.5.5 `ns_map spot::tgba_explicit::name_state_map_` [protected]

7.42.5.6 `bdd spot::tgba_explicit::neg_acceptance_conditions_` [protected]

7.42.5.7 `sn_map spot::tgba_explicit::state_name_map_` [protected]

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

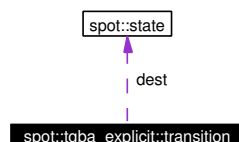
- [tgbaexplicit.hh](#)

7.43 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`

7.43.1 Detailed Description

Explicit transitions (used by [spot::tgba_explicit](#)).

7.43.2 Member Data Documentation

7.43.2.1 bdd [spot::tgba_explicit::transition::acceptance_conditions](#)

7.43.2.2 bdd [spot::tgba_explicit::transition::condition](#)

7.43.2.3 state* [spot::tgba_explicit::transition::dest](#)

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

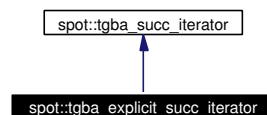
- [tgbaexplicit.hh](#)

7.44 spot::tgba_explicit_succ_iterator Class Reference

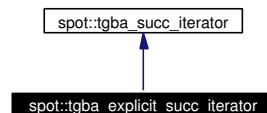
Successor iterators used by [spot::tgba_explicit](#).

```
#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_](#)

7.44.1 Detailed Description

Successor iterators used by [spot::tgba_explicit](#).

7.44.2 Constructor & Destructor Documentation

7.44.2.1 [spot::tgba_explicit_succ_iterator::tgba_explicit_succ_iterator](#) (const [tgba_explicit::state](#) * [s](#), bdd [all_acc](#))

7.44.2.2 virtual [spot::tgba_explicit_succ_iterator::~~tgba_explicit_succ_iterator](#) () [inline, virtual]

7.44.3 Member Function Documentation

7.44.3.1 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](#).

7.44.3.2 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](#).

7.44.3.3 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](#).

7.44.3.4 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](#).

7.44.3.5 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](#).

7.44.3.6 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](#).

7.44.4 Member Data Documentation**7.44.4.1 bdd spot::tgba_explicit_succ_iterator::all_acceptance_conditions_ [private]****7.44.4.2 tgba_explicit::state::const_iterator spot::tgba_explicit_succ_iterator::i_ [private]****7.44.4.3 const tgba_explicit::state* spot::tgba_explicit_succ_iterator::s_ [private]**

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

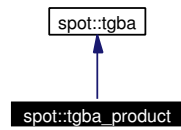
- [tgbaexplicit.hh](#)

7.45 spot::tgba_product Class Reference

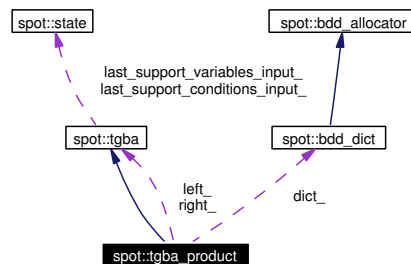
A lazy product. (States are computed on the fly.).

```
#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_](#)

7.45.1 Detailed Description

A lazy product. (States are computed on the fly.).

7.45.2 Constructor & Destructor Documentation

7.45.2.1 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.

7.45.2.2 virtual spot::tgba_product::~~tgba_product () [virtual]

7.45.2.3 spot::tgba_product::tgba_product (const [tgba_product](#) &) [private]

7.45.3 Member Function Documentation

7.45.3.1 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](#).

7.45.3.2 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](#).

7.45.3.3 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](#).

7.45.3.4 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](#).

7.45.3.5 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](#).

7.45.3.6 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](#).

7.45.3.7 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`.

7.45.3.8 `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 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`.

7.45.3.9 `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 *local_state*, *global_state* is not adopted by `succ_iter`.

global_automaton In a product, the state of the global product automaton. Otherwise, 0.

Implements `spot::tgba`.

7.45.3.10 `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.

7.45.3.11 `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.

7.45.3.12 `tgba_product& spot::tgba_product::tgba_product::operator= (const tgba_product &)` [private]

7.45.4 Member Data Documentation

7.45.4.1 `bdd spot::tgba_product::all_acceptance_conditions_` [private]

7.45.4.2 `bdd_dict* spot::tgba_product::dict_` [private]

7.45.4.3 `const tgba* spot::tgba_product::left_` [private]

7.45.4.4 `bdd spot::tgba_product::left_acc_complement_` [private]

7.45.4.5 `bdd spot::tgba_product::neg_acceptance_conditions_` [private]

7.45.4.6 `const tgba* spot::tgba_product::right_` [private]

7.45.4.7 `bdd spot::tgba_product::right_acc_complement_` [private]

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

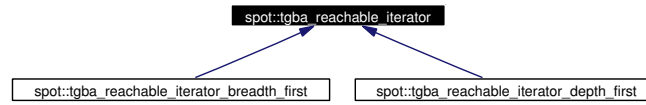
- [tgbaproduct.hh](#)

7.46 `spot::tgba_reachable_iterator` Class Reference

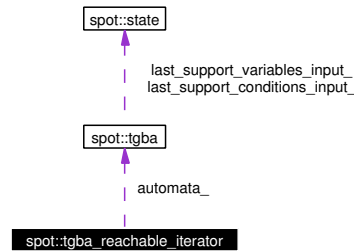
Iterate over all reachable states of a `spot::tgba`.

```
#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.

7.46.1 Detailed Description

Iterate over all reachable states of a [spot::tgba](#).

7.46.2 Member Typedef Documentation

7.46.2.1 typedef [Sgi::hash_map](#)<const [state](#)*, int, [state_ptr_hash](#), [state_ptr_equal](#)> [spot::tgba_reachable_iterator::seen_map](#) [protected]

7.46.3 Constructor & Destructor Documentation

7.46.3.1 [spot::tgba_reachable_iterator::tgba_reachable_iterator](#) (const [tgba](#) * *a*)

7.46.3.2 virtual [spot::tgba_reachable_iterator::~~tgba_reachable_iterator](#) () [virtual]

7.46.4 Member Function Documentation

7.46.4.1 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](#).

7.46.4.2 virtual void [spot::tgba_reachable_iterator::end](#) () [virtual]

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

7.46.4.3 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](#).

7.46.4.4 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.

7.46.4.5 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*.

7.46.4.6 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.

7.46.4.7 virtual void spot::tgba_reachable_iterator::start () [virtual]

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

7.46.5 Member Data Documentation

7.46.5.1 const [tgba](#)* spot::tgba_reachable_iterator::automata_ [protected]

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

7.46.5.2 [seen_map](#) spot::tgba_reachable_iterator::seen [protected]

States already seen.

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

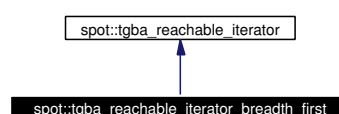
- [reachiter.hh](#)

7.47 spot::tgba_reachable_iterator_breadth_first Class Reference

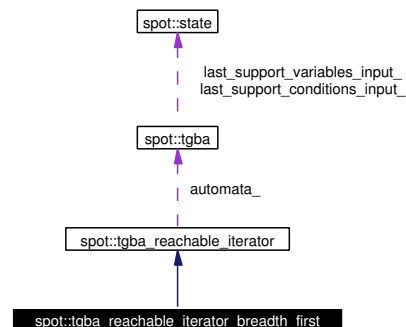
An implementation of [spot::tgba_reachable_iterator](#) that browses states breadth first.

```
#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 state yet to explore.
- const [tgba](#) * [automata_](#)
The [spot::tgba](#) to explore.
- [seen_map](#) [seen](#)
States already seen.

7.47.1 Detailed Description

An implementation of [spot::tgba_reachable_iterator](#) that browses states breadth first.

7.47.2 Member Typedef Documentation

7.47.2.1 `typedef Sgi::hash_map<const state*, int, state_ptr_hash, state_ptr_equal> spot::tgba_reachable_iterator::seen_map [protected, inherited]`

7.47.3 Constructor & Destructor Documentation

7.47.3.1 `spot::tgba_reachable_iterator_breadth_first::tgba_reachable_iterator_breadth_first (const tgba * a)`

7.47.4 Member Function Documentation

7.47.4.1 `virtual void spot::tgba_reachable_iterator_breadth_first::add_state (const state * s) [virtual]`

Implements [spot::tgba_reachable_iterator](#).

7.47.4.2 `virtual void spot::tgba_reachable_iterator::end () [virtual, inherited]`

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

7.47.4.3 `virtual const state* spot::tgba_reachable_iterator_breadth_first::next_state () [virtual]`

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

Implements [spot::tgba_reachable_iterator](#).

7.47.4.4 `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.

7.47.4.5 `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*.

7.47.4.6 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.

7.47.4.7 virtual void spot::tgba_reachable_iterator::start () [virtual, inherited]

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

7.47.5 Member Data Documentation**7.47.5.1 const tgba* spot::tgba_reachable_iterator::automata_ [protected, inherited]**

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

7.47.5.2 seen_map spot::tgba_reachable_iterator::seen [protected, inherited]

States already seen.

7.47.5.3 std::deque<const state*> spot::tgba_reachable_iterator_breadth_first::todo [protected]

A queue of state yet to explore.

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

- [reachiter.hh](#)

7.48 spot::tgba_reachable_iterator_depth_first Class Reference

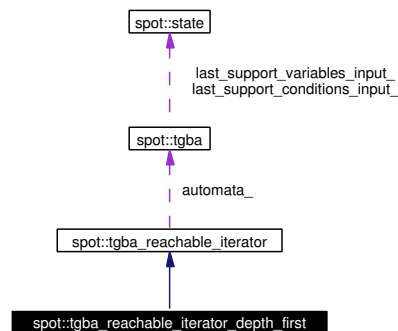
An implementation of [spot::tgba_reachable_iterator](#) that browses states depth first.

```
#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 state yet to explore.
- const [tgba](#) * [automata_](#)
The [spot::tgba](#) to explore.
- [seen_map](#) [seen](#)
States already seen.

7.48.1 Detailed Description

An implementation of [spot::tgba_reachable_iterator](#) that browses states depth first.

7.48.2 Member Typedef Documentation

7.48.2.1 `typedef Sgi::hash_map<const state*, int, state_ptr_hash, state_ptr_equal> spot::tgba_reachable_iterator::seen_map [protected, inherited]`

7.48.3 Constructor & Destructor Documentation

7.48.3.1 `spot::tgba_reachable_iterator_depth_first::tgba_reachable_iterator_depth_first (const tgba * a)`

7.48.4 Member Function Documentation

7.48.4.1 `virtual void spot::tgba_reachable_iterator_depth_first::add_state (const state * s) [virtual]`

Implements [spot::tgba_reachable_iterator](#).

7.48.4.2 `virtual void spot::tgba_reachable_iterator::end () [virtual, inherited]`

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

7.48.4.3 `virtual const state* spot::tgba_reachable_iterator_depth_first::next_state () [virtual]`

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

Implements [spot::tgba_reachable_iterator](#).

7.48.4.4 `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.

7.48.4.5 `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*.

7.48.4.6 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.

7.48.4.7 virtual void spot::tgba_reachable_iterator::start () [virtual, inherited]

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

7.48.5 Member Data Documentation**7.48.5.1 const tgba* spot::tgba_reachable_iterator::automata_** [protected, inherited]

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

7.48.5.2 seen_map spot::tgba_reachable_iterator::seen [protected, inherited]

States already seen.

7.48.5.3 std::stack<const state*> spot::tgba_reachable_iterator_depth_first::todo [protected]

A stack of state yet to explore.

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

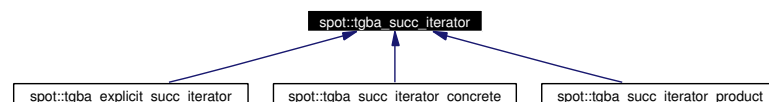
- [reachiter.hh](#)

7.49 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.

7.49.1 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.

7.49.2 Constructor & Destructor Documentation

7.49.2.1 virtual `spot::tgba_succ_iterator::~~tgba_succ_iterator ()` [inline, virtual]

7.49.3 Member Function Documentation

7.49.3.1 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`.

7.49.3.2 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`.

7.49.3.3 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](#).

7.49.3.4 `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](#).

7.49.3.5 `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](#).

7.49.3.6 `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:

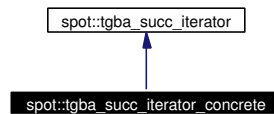
- [succiter.hh](#)

7.50 `spot::tgba_succ_iterator_concrete` Class Reference

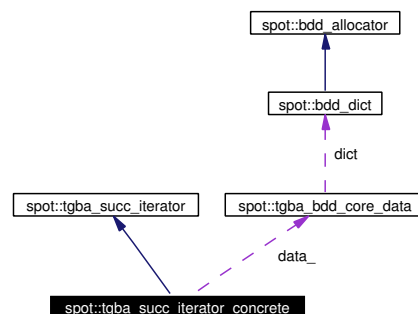
A concrete iterator over successors of a TGBA state.

```
#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.

7.50.1 Detailed Description

A concrete iterator over successors of a TGBA state.

7.50.2 Constructor & Destructor Documentation

7.50.2.1 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.

7.50.2.2 virtual spot::tgba_succ_iterator_concrete::~~tgba_succ_iterator_concrete () [virtual]

7.50.3 Member Function Documentation

7.50.3.1 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](#).

7.50.3.2 `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](#).

7.50.3.3 `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](#).

7.50.3.4 `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](#).

7.50.3.5 `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](#).

7.50.3.6 `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](#).

7.50.4 Member Data Documentation**7.50.4.1** `bdd spot::tgba_succ_iterator_concrete::current_` [private]

Current successor, as a conjunction of atomic proposition and Next variables.

7.50.4.2 bdd [spot::tgba_succ_iterator_concrete::current_acc_](#) [private]

Accepting conditions for the current transition.

7.50.4.3 bdd [spot::tgba_succ_iterator_concrete::current_state_](#) [private]

Current successor, as a conjunction of Now variables.

7.50.4.4 const [tgba_bdd_core_data&](#) [spot::tgba_succ_iterator_concrete::data_](#) [private]

Core data of the automaton.

7.50.4.5 bdd [spot::tgba_succ_iterator_concrete::succ_set_](#) [private]

The set of successors.

7.50.4.6 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:

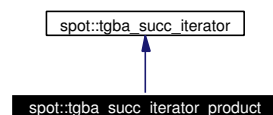
- [succiterconcrete.hh](#)

7.51 spot::tgba_succ_iterator_product Class Reference

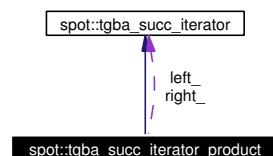
Iterate over the successors of a product computed on the fly.

```
#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_](#) ()

7.51.1 Detailed Description

Iterate over the successors of a product computed on the fly.

7.51.2 Constructor & Destructor Documentation

7.51.2.1 spot::tgba_succ_iterator_product::tgba_succ_iterator_product ([tgba_succ_iterator](#) * *left*, [tgba_succ_iterator](#) * *right*, bdd *left_neg*, bdd *right_neg*)

7.51.2.2 virtual spot::tgba_succ_iterator_product::~[tgba_succ_iterator_product](#) () [virtual]

7.51.3 Member Function Documentation

7.51.3.1 `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](#).

7.51.3.2 `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](#).

7.51.3.3 `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](#).

7.51.3.4 `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](#).

7.51.3.5 `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](#).

7.51.3.6 `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`.

7.51.3.7 `void spot::tgba_succ_iterator_product::next_non_false_ ()` [private]

7.51.3.8 `void spot::tgba_succ_iterator_product::step_ ()` [private]

Internal routines to advance to the next successor.

7.51.4 Member Data Documentation

7.51.4.1 `bdd spot::tgba_succ_iterator_product::current_cond_` [protected]

7.51.4.2 `tgba_succ_iterator* spot::tgba_succ_iterator_product::left_` [protected]

7.51.4.3 `bdd spot::tgba_succ_iterator_product::left_neg_` [protected]

7.51.4.4 `tgba_succ_iterator* spot::tgba_succ_iterator_product::right_` [protected]

7.51.4.5 `bdd spot::tgba_succ_iterator_product::right_neg_` [protected]

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

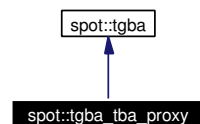
- `tgbaproduct.hh`

7.52 spot::tgba_tba_proxy Class Reference

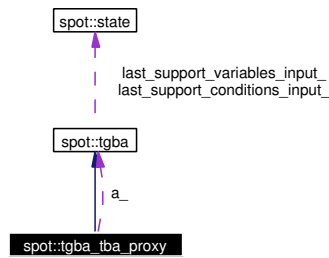
Degeneralize a `spot::tgba` on the fly.

```
#include <tgbatba.hh>
```

Inheritance diagram for `spot::tgba_tba_proxy`:



Collaboration diagram for `spot::tgba_tba_proxy`:



Public 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
- [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 Types

- typedef std::map< bdd, bdd, `bdd_less_than` > `cycle_map`

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_map` `acc_cycle_`
- bdd `the_acceptance_cond_`

7.52.1 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.

7.52.2 Member Typedef Documentation

7.52.2.1 typedef std::map< bdd, bdd, `bdd_less_than` > `spot::tgba_tba_proxy::cycle_map` [private]

7.52.3 Constructor & Destructor Documentation

7.52.3.1 `spot::tgba_tba_proxy::tgba_tba_proxy` (const `tgba * a`)

7.52.3.2 virtual `spot::tgba_tba_proxy::~~tgba_tba_proxy` () [virtual]

7.52.3.3 `spot::tgba_tba_proxy::tgba_tba_proxy` (const `tgba_tba_proxy` &) [private]

7.52.4 Member Function Documentation

7.52.4.1 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](#).

7.52.4.2 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](#).

7.52.4.3 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](#).

7.52.4.4 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](#).

7.52.4.5 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](#).

7.52.4.6 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](#).

7.52.4.7 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](#).

7.52.4.8 `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](#).

7.52.4.9 `bool spot::tgba_tba_proxy::state_is_accepting (const state * state) const`

7.52.4.10 `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 state of the global product automaton. Otherwise, 0.

Implements [spot::tgba](#).

7.52.4.11 `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.

7.52.4.12 `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.

7.52.4.13 `tgba_tba_proxy& spot::tgba_tba_proxy::tgba_tba_proxy::operator= (const tgba_tba_proxy &) [private]`

7.52.5 Member Data Documentation

7.52.5.1 `const tgba* spot::tgba_tba_proxy::a_ [private]`

7.52.5.2 `cycle_map spot::tgba_tba_proxy::acc_cycle_ [private]`

7.52.5.3 `bdd spot::tgba_tba_proxy::the_acceptance_cond_ [private]`

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

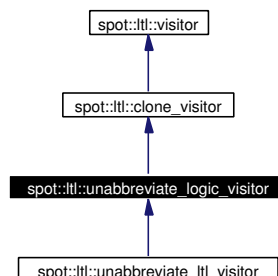
- [tgbatba.hh](#)

7.53 spot::ltl::unabbreviate_logic_visitor Class Reference

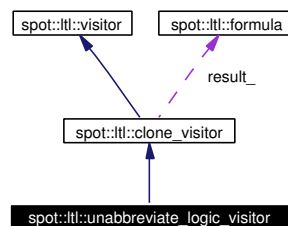
Clone and rewrite a formula to remove most of the abbreviated logical operators.

```
#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`

7.53.1 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.

7.53.2 Member Typedef Documentation

7.53.2.1 `typedef clone_visitor spot::ltl::unabbreviate_logic_visitor::super` [private]

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

7.53.3 Constructor & Destructor Documentation

7.53.3.1 `spot::ltl::unabbreviate_logic_visitor::unabbreviate_logic_visitor()`

7.53.3.2 virtual spot::ltl::unabbreviate_logic_visitor::~unabbreviate_logic_visitor ()
[virtual]

7.53.4 Member Function Documentation

7.53.4.1 virtual formula* spot::ltl::unabbreviate_logic_visitor::recurse (formula *f) [virtual]

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

Reimplemented in [spot::ltl::unabbreviate_ltl_visitor](#).

7.53.4.2 formula* spot::ltl::clone_visitor::result () const [inherited]

7.53.4.3 void spot::ltl::clone_visitor::visit (constant *c)

7.53.4.4 void spot::ltl::clone_visitor::visit (multop *mo)

7.53.4.5 void spot::ltl::clone_visitor::visit (binop *bo)

7.53.4.6 void spot::ltl::clone_visitor::visit (unop *uo)

7.53.4.7 void spot::ltl::clone_visitor::visit (atomic_prop *ap)

7.53.4.8 void spot::ltl::unabbreviate_logic_visitor::visit (binop *bo) [virtual]

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

7.53.5 Member Data Documentation

7.53.5.1 formula* spot::ltl::clone_visitor::result_ [protected, inherited]

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

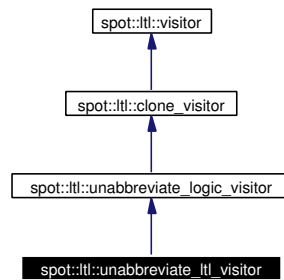
- [lunabbrev.hh](#)

7.54 spot::ltl::unabbreviate_ltl_visitor Class Reference

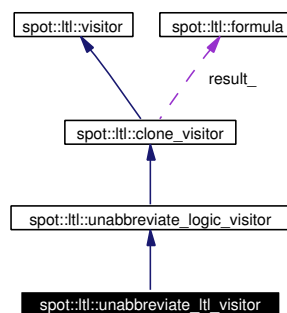
Clone and rewrite a formula to remove most of the abbreviated LTL and logical operators.

```
#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](#)

7.54.1 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.

7.54.2 Member Typedef Documentation

7.54.2.1 `typedef unabbreviate_logic_visitor spot::ltl::unabbreviate_ltl_visitor::super [private]`

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

7.54.3 Constructor & Destructor Documentation

7.54.3.1 `spot::ltl::unabbreviate_ltl_visitor::unabbreviate_ltl_visitor ()`

7.54.3.2 `virtual spot::ltl::unabbreviate_ltl_visitor::~~unabbreviate_ltl_visitor () [virtual]`

7.54.4 Member Function Documentation

7.54.4.1 `formula* spot::ltl::unabbreviate_ltl_visitor::recurse (formula *f) [virtual]`

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

7.54.4.2 `formula* spot::ltl::clone_visitor::result () const [inherited]`

7.54.4.3 `void spot::ltl::clone_visitor::visit (constant *c) [inherited]`

7.54.4.4 `void spot::ltl::clone_visitor::visit (multop *mo) [inherited]`

7.54.4.5 `void spot::ltl::clone_visitor::visit (atomic_prop *ap) [inherited]`

7.54.4.6 `void spot::ltl::unabbreviate_logic_visitor::visit (binop *bo) [virtual, inherited]`

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

7.54.4.7 `void spot::ltl::unabbreviate_ltl_visitor::visit (unop *uo) [virtual]`

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

7.54.5 Member Data Documentation

7.54.5.1 `formula* spot::ltl::clone_visitor::result_ [protected, inherited]`

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

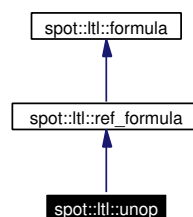
- [tunabbrev.hh](#)

7.55 spot::ltl::unop Class Reference

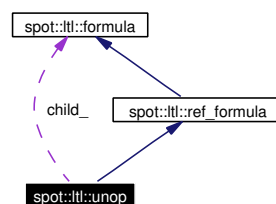
Unary operator.

```
#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).

Static Protected Attributes

- `map instances`

Private Attributes

- `type op_`
- `formula * child_`

7.55.1 Detailed Description

Unary operator.

7.55.2 Member Typedef Documentation

7.55.2.1 typedef std::map<pair, formula*> spot::ltl::unop::map [protected]

7.55.2.2 typedef std::pair<type, formula*> spot::ltl::unop::pair [protected]

7.55.3 Member Enumeration Documentation

7.55.3.1 enum spot::ltl::unop::type

Enumeration values:

Not

X

F

G

7.55.4 Constructor & Destructor Documentation

7.55.4.1 spot::ltl::unop::unop (type op, formula * child) [protected]

7.55.4.2 virtual spot::ltl::unop::~~unop () [protected, virtual]

7.55.5 Member Function Documentation

7.55.5.1 virtual void spot::ltl::unop::accept (const_visitor & v) const [virtual]

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

Implements spot::ltl::formula.

7.55.5.2 virtual void spot::ltl::unop::accept (visitor & v) [virtual]

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

Implements spot::ltl::formula.

7.55.5.3 formula* spot::ltl::unop::child ()

Get the sole operand of this operator.

7.55.5.4 const formula* spot::ltl::unop::child () const

Get the sole operand of this operator.

7.55.5.5 unop* spot::ltl::unop::instance (type op, formula * child) [static]

Build an unary operator with operation *op* and child *child*.

7.55.5.6 unsigned spot::ltl::unop::instance_count () [static]

Number of instantiated unary operators. For debugging.

7.55.5.7 `type spot::ltl::unop::op () const`

Get the type of this operator.

7.55.5.8 `const char* spot::ltl::unop::op_name () const`

Get the type of this operator, as a string.

7.55.5.9 `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.

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

increment reference counter if any

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

7.55.5.11 `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.

7.55.5.12 `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`.

7.55.6 Member Data Documentation**7.55.6.1** `formula* spot::ltl::unop::child_ [private]`**7.55.6.2** `map spot::ltl::unop::instances [static, protected]`**7.55.6.3** `type spot::ltl::unop::op_ [private]`

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

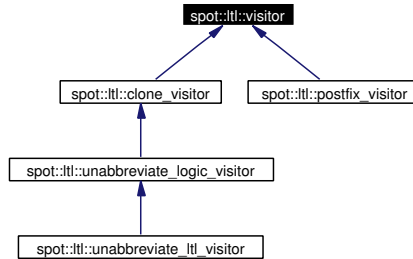
- `unop.hh`

7.56 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

7.56.1 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.

7.56.2 Member Function Documentation

7.56.2.1 virtual void spot::ltl::visitor::visit ([multop](#) * node) [pure virtual]

Implemented in [spot::ltl::clone_visitor](#), and [spot::ltl::postfix_visitor](#).

7.56.2.2 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](#).

7.56.2.3 virtual void spot::ltl::visitor::visit ([binop](#) * node) [pure virtual]

Implemented in [spot::ltl::clone_visitor](#), [spot::ltl::unabbreviate_logic_visitor](#), and [spot::ltl::postfix_visitor](#).

7.56.2.4 virtual void spot::ltl::visitor::visit ([constant](#) * node) [pure virtual]

Implemented in [spot::ltl::clone_visitor](#), and [spot::ltl::postfix_visitor](#).

7.56.2.5 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:

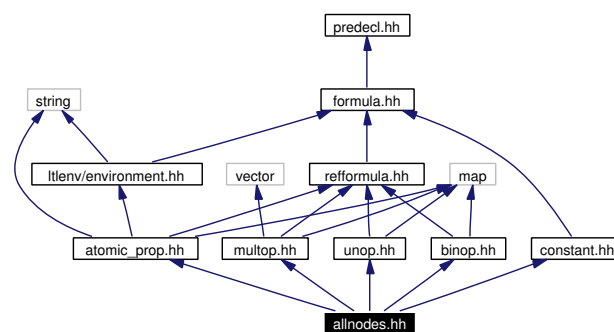
- [visitor.hh](#)

8 spot File Documentation

8.1 allnodes.hh File Reference

```
#include "binop.hh"
#include "unop.hh"
#include "multop.hh"
#include "atomic_prop.hh"
#include "constant.hh"
```

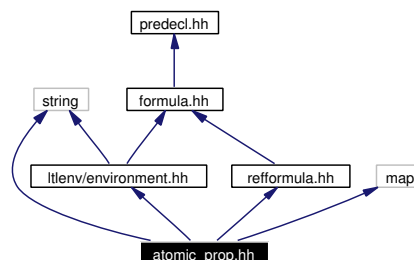
Include dependency graph for allnodes.hh:



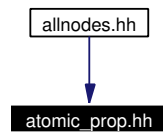
8.2 atomic_prop.hh File Reference

```
#include <string>
#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

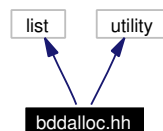
- namespace `spot`
- namespace `spot::ltl`

8.3 bddalloc.hh File Reference

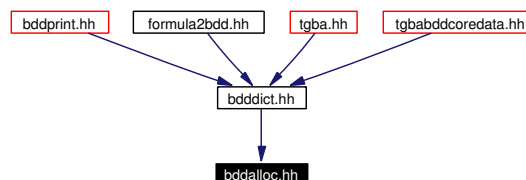
```
#include <list>
```

```
#include <utility>
```

Include dependency graph for bddalloc.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace `spot`

8.4 bdddict.hh File Reference

```
#include "misc/hash.hh"
```

```
#include <list>
```

```
#include <set>
```

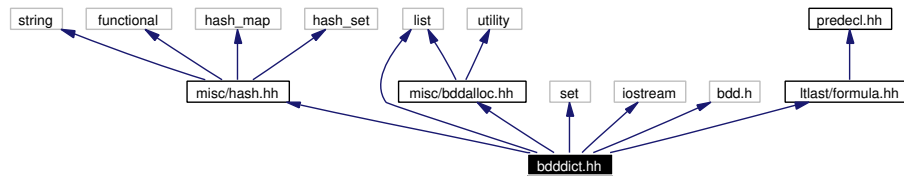
```
#include <iostream>
```

```
#include <bdd.h>
```

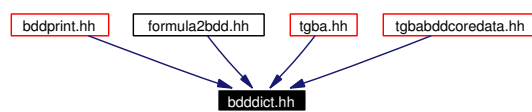
```
#include "ltlast/formula.hh"
```

```
#include "misc/bddalloc.hh"
```

Include dependency graph for bdddict.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

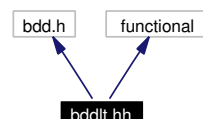
- namespace [spot](#)

8.5 bddlt.hh File Reference

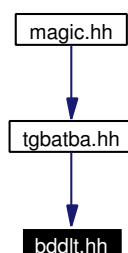
```
#include <bdd.h>
```

```
#include <functional>
```

Include dependency graph for bddlt.hh:



This graph shows which files directly or indirectly include this file:



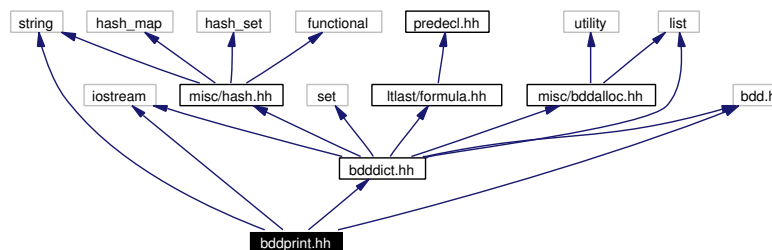
Namespaces

- namespace [spot](#)

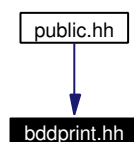
8.6 bddprint.hh File Reference

```
#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:



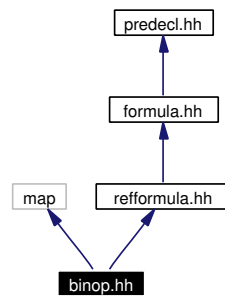
Namespaces

- namespace [spot](#)

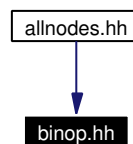
8.7 binop.hh File Reference

```
#include <map>
#include "reformula.hh"
```

Include dependency graph for binop.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

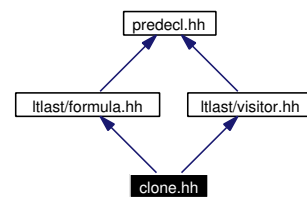
- namespace `spot`
- namespace `spot::ltl`

8.8 clone.hh File Reference

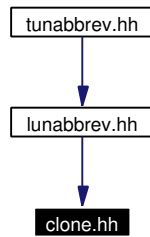
```
#include "ltlast/formula.hh"
```

```
#include "ltlast/visitor.hh"
```

Include dependency graph for `clone.hh`:



This graph shows which files directly or indirectly include this file:



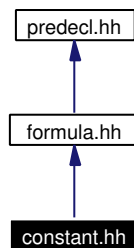
Namespaces

- namespace `spot`
- namespace `spot::ltl`

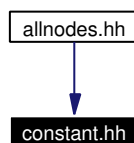
8.9 constant.hh File Reference

```
#include "formula.hh"
```

Include dependency graph for constant.hh:



This graph shows which files directly or indirectly include this file:



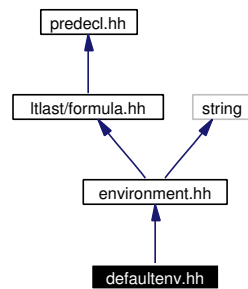
Namespaces

- namespace `spot`
- namespace `spot::ltl`

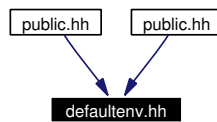
8.10 defaultenv.hh File Reference

```
#include "environment.hh"
```

Include dependency graph for defaultenv.hh:



This graph shows which files directly or indirectly include this file:



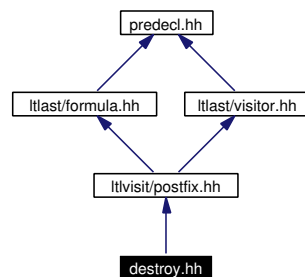
Namespaces

- namespace `spot`
- namespace `spot::ltl`

8.11 destroy.hh File Reference

```
#include "ltlvisit/postfix.hh"
```

Include dependency graph for `destroy.hh`:



Namespaces

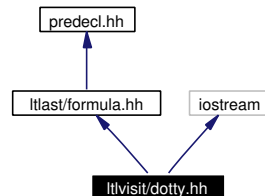
- namespace `spot`
- namespace `spot::ltl`

8.12 dotty.hh File Reference

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

```
#include <iostream>
```

Include dependency graph for `ltlvisit/dotty.hh`:



Namespaces

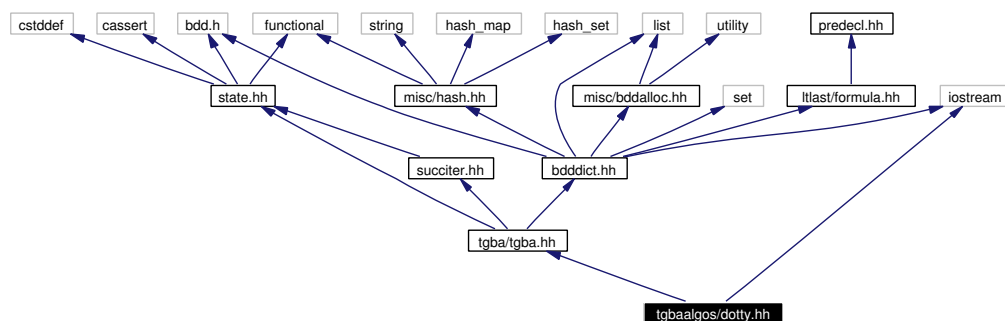
- namespace `spot`
- namespace `spot::ltl`

8.13 **dotty.hh** File Reference

```
#include "tgba/tgba.hh"
```

```
#include <iostream>
```

Include dependency graph for `tgbaalgos/dotty.hh`:



Namespaces

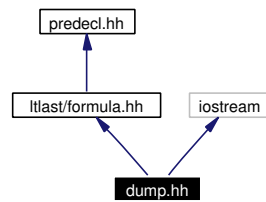
- namespace `spot`

8.14 **dump.hh** File Reference

```
#include "ltlast/formula.hh"
```

```
#include <iostream>
```

Include dependency graph for `dump.hh`:



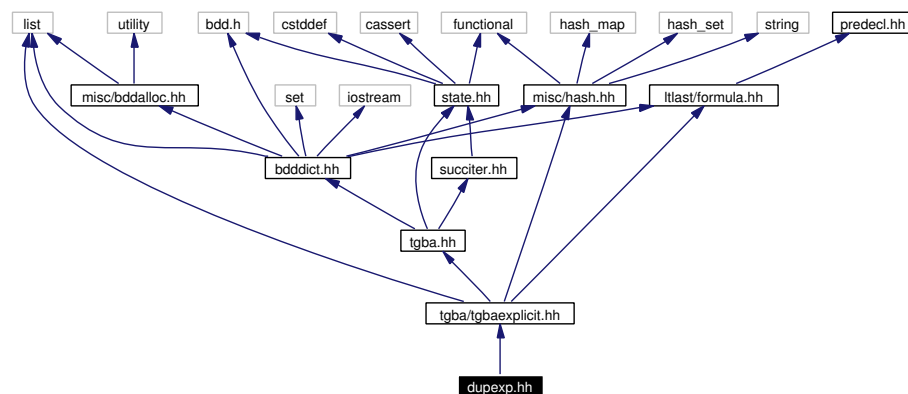
Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

8.15 dupexp.hh File Reference

```
#include "tgba/tgbaexplicit.hh"
```

Include dependency graph for dupexp.hh:



Namespaces

- namespace [spot](#)

8.16 emptinesscheck.hh File Reference

```
#include "tgba/tgba.hh"
```

```
#include "misc/hash.hh"
```

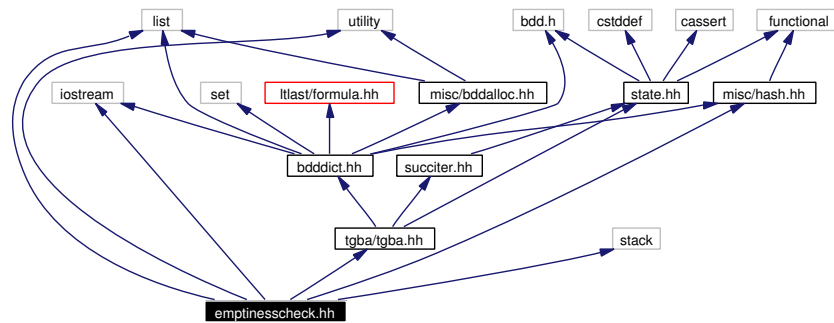
```
#include <stack>
```

```
#include <list>
```

```
#include <utility>
```

```
#include <iostream>
```

Include dependency graph for emptinesscheck.hh:



Namespaces

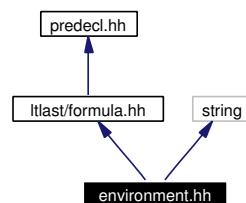
- namespace [spot](#)

8.17 environment.hh File Reference

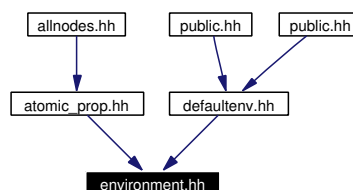
```
#include "ltlast/formula.hh"
```

```
#include <string>
```

Include dependency graph for environment.hh:



This graph shows which files directly or indirectly include this file:



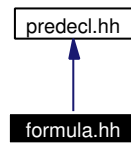
Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

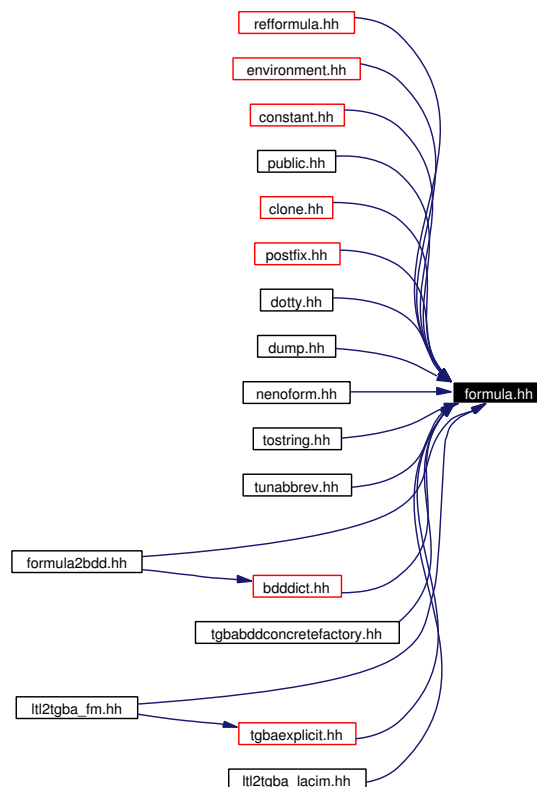
8.18 formula.hh File Reference

```
#include "predecl.hh"
```

Include dependency graph for formula.hh:



This graph shows which files directly or indirectly include this file:



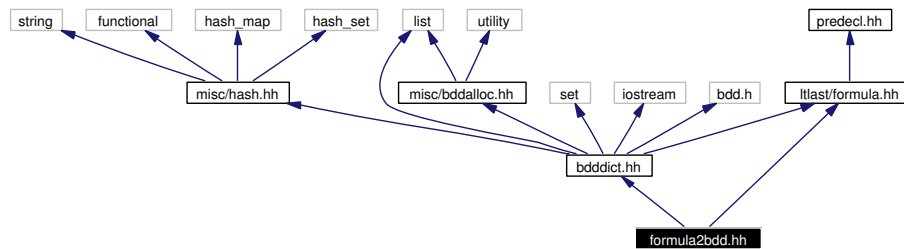
Namespaces

- namespace `spot`
- namespace `spot::ltn`

8.19 formula2bdd.hh File Reference

```
#include "bdddict.hh"
#include "ltlast/formula.hh"
```

Include dependency graph for formula2bdd.hh:



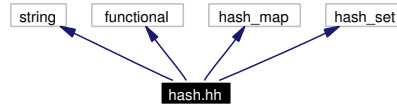
Namespaces

- namespace **spot**

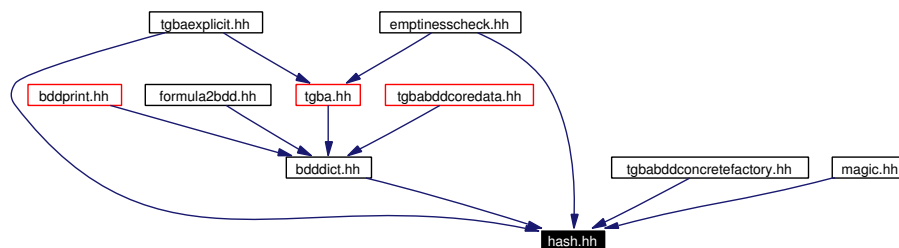
8.20 hash.hh File Reference

```
#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

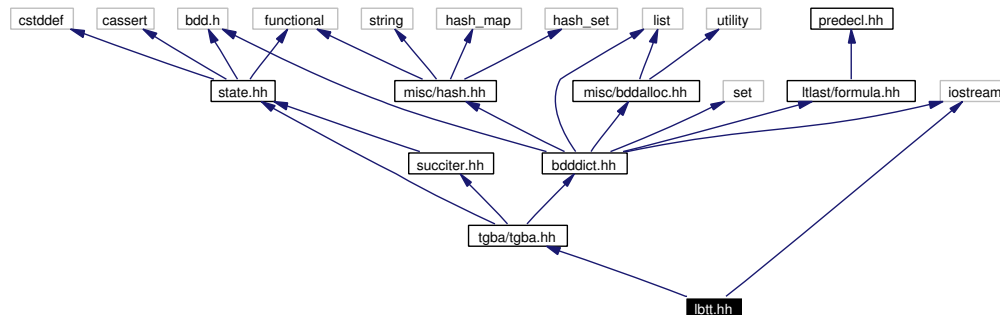
- namespace **spot**

8.21 lbtt.hh File Reference

```
#include "tgba/tgba.hh"
```

```
#include <iostream>
```

Include dependency graph for lbtt.hh:



Namespaces

- namespace [spot](#)

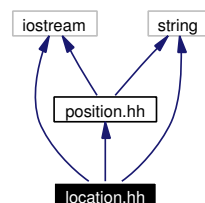
8.22 location.hh File Reference

```
#include <iostream>
```

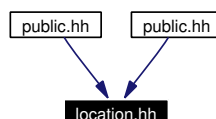
```
#include <string>
```

```
#include "position.hh"
```

Include dependency graph for location.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [yy](#)

8.22.1 Detailed Description

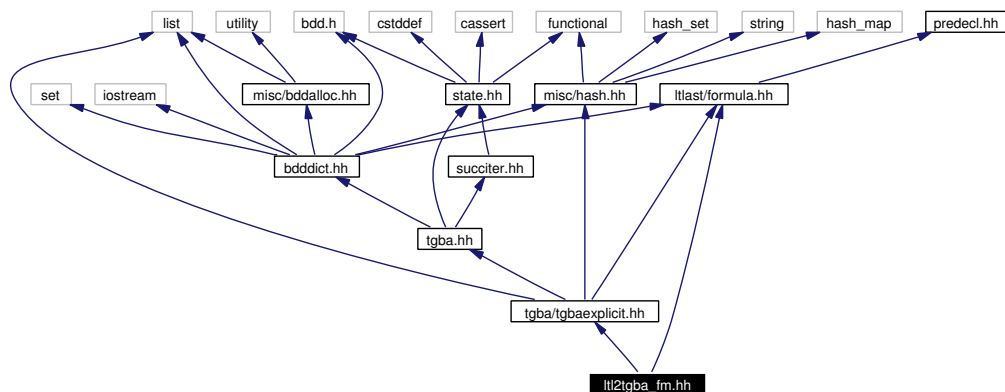
Define the Location class.

8.23 ltl2tgba_fm.hh File Reference

```
#include "ltlast/formula.hh"
```

```
#include "tgba/tgbaexplicit.hh"
```

Include dependency graph for ltl2tgba_fm.hh:



Namespaces

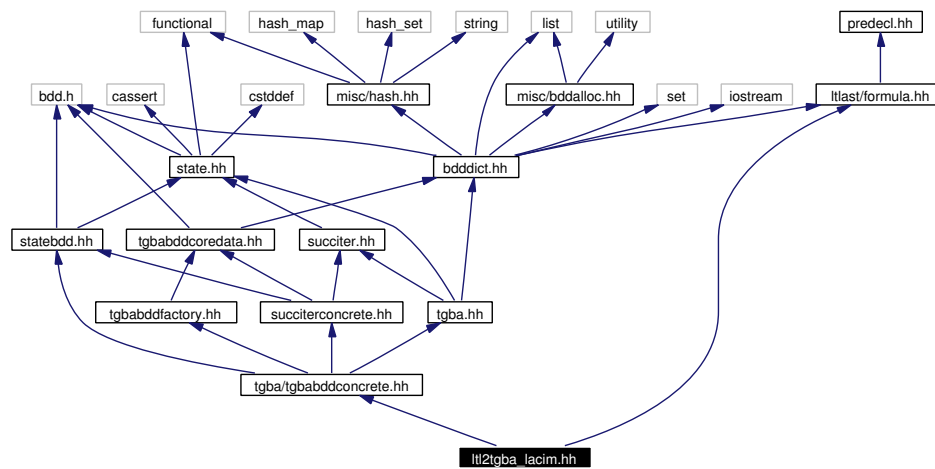
- namespace **spot**

8.24 ltl2tgba_lacim.hh File Reference

```
#include "ltlast/formula.hh"
```

```
#include "tgba/tgbabddconcrete.hh"
```

Include dependency graph for ltl2tgba_lacim.hh:



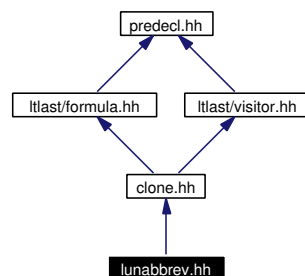
Namespaces

- namespace **spot**

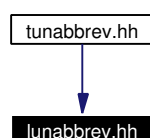
8.25 lunabbrev.hh File Reference

```
#include "clone.hh"
```

Include dependency graph for lunabbrev.hh:



This graph shows which files directly or indirectly include this file:



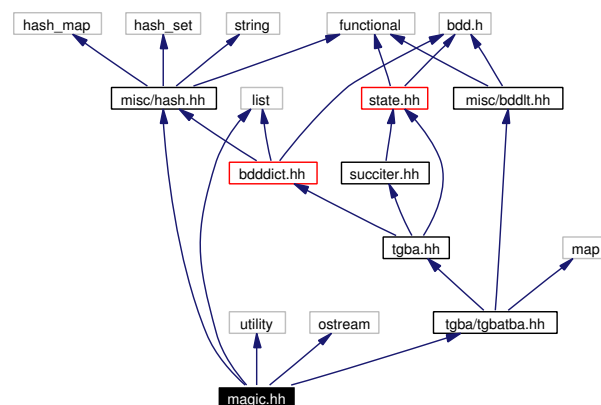
Namespaces

- namespace `spot`
- namespace `spot::ltn`

8.26 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

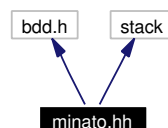
- namespace [spot](#)

8.27 mainpage.dox File Reference

8.28 minato.hh File Reference

```
#include <bdd.h>
#include <stack>
```

Include dependency graph for minato.hh:



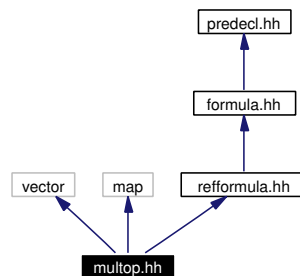
Namespaces

- namespace [spot](#)

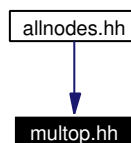
8.29 multop.hh File Reference

```
#include <vector>
#include <map>
#include "reformula.hh"
```

Include dependency graph for multop.hh:



This graph shows which files directly or indirectly include this file:



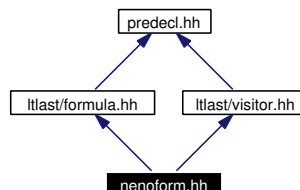
Namespaces

- namespace `spot`
- namespace `spot::ltl`

8.30 nenoform.hh File Reference

```
#include "ltlast/formula.hh"
#include "ltlast/visitor.hh"
```

Include dependency graph for nenoform.hh:



Namespaces

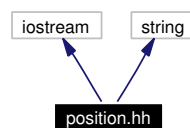
- namespace [spot](#)
- namespace [spot::ltd](#)

8.31 position.hh File Reference

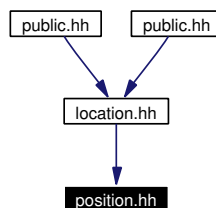
```
#include <iostream>
```

```
#include <string>
```

Include dependency graph for position.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [yy](#)

8.31.1 Detailed Description

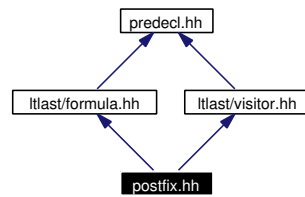
Define the Location class.

8.32 postfix.hh File Reference

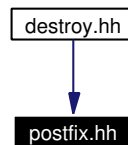
```
#include "ltdlast/formula.hh"
```

```
#include "ltdlast/visitor.hh"
```

Include dependency graph for postfix.hh:



This graph shows which files directly or indirectly include this file:

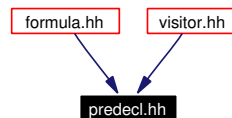


Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

8.33 predecl.hh File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

8.34 public.hh File Reference

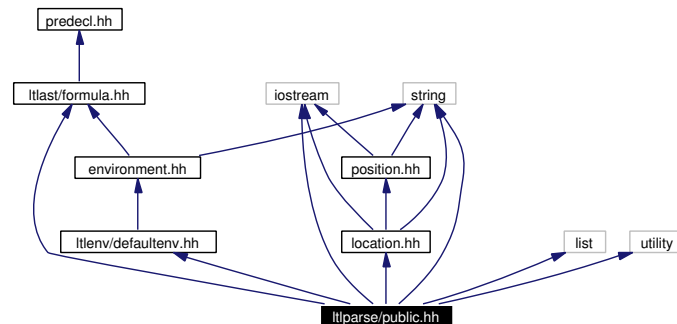
```

#include "ltlast/formula.hh"
#include "location.hh"
#include "ltlenv/defaultenv.hh"
#include <string>
#include <list>
#include <utility>

```

```
#include <iostream>
```

Include dependency graph for ltlparse/public.hh:



Namespaces

- namespace `spot`
- namespace `spot::ltl`

8.35 public.hh File Reference

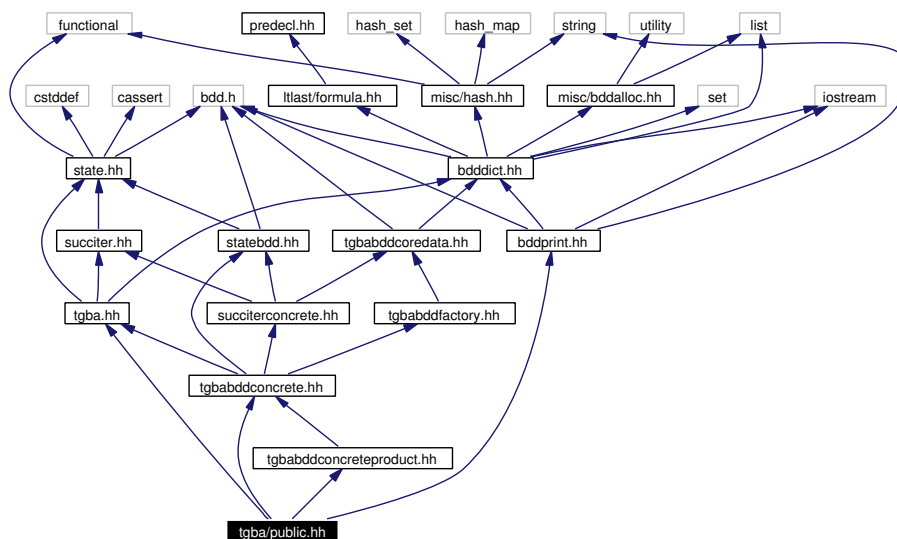
```
#include "tgba.hh"
```

```
#include "tgbabddconcrete.hh"
```

```
#include "tgbabddconcreteproduct.hh"
```

```
#include "bddprint.hh"
```

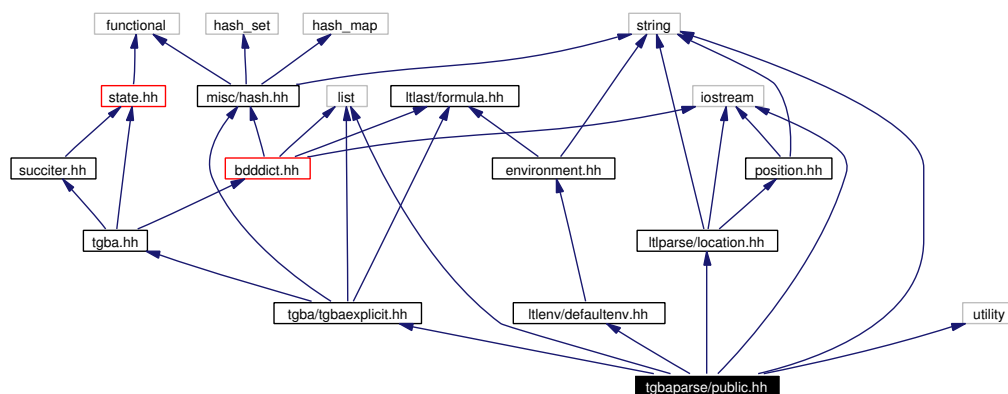
Include dependency graph for tgba/public.hh:



8.36 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 `tgba/tgbaexplicit/public.hh`:



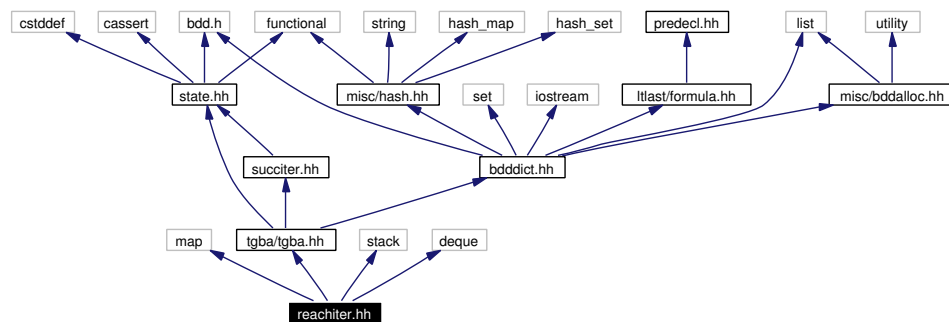
Namespaces

- namespace `spot`

8.37 reachiter.hh File Reference

```
#include <map>
#include "tgba/tgba.hh"
#include <stack>
#include <deque>
```

Include dependency graph for `reachiter.hh`:



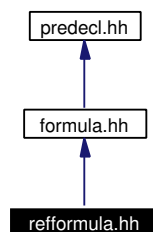
Namespaces

- namespace **spot**

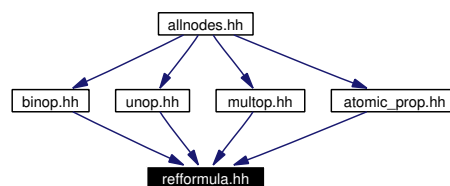
8.38 reformula.hh File Reference

```
#include "formula.hh"
```

Include dependency graph for refformula.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

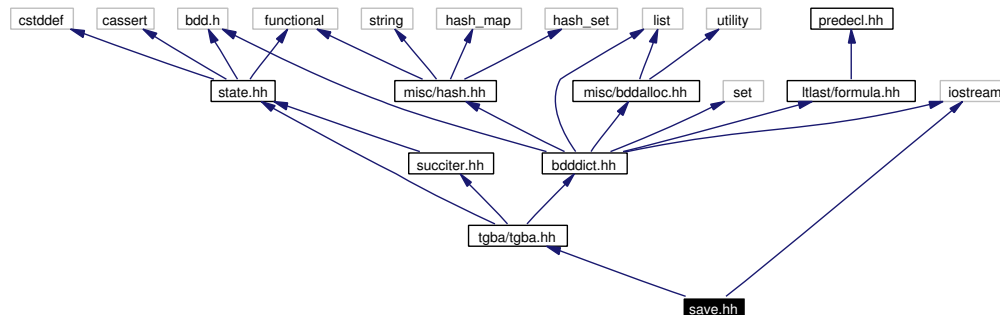
- namespace `spot`
- namespace `spot::ltrl`

8.39 save.hh File Reference

```
#include "tgba/tgba.hh"
```

```
#include <iostream>
```

Include dependency graph for save.hh:



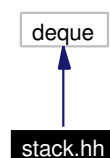
Namespaces

- namespace [spot](#)

8.40 stack.hh File Reference

```
#include <deque>
```

Include dependency graph for stack.hh:



Namespaces

- namespace [yy](#)

8.41 state.hh File Reference

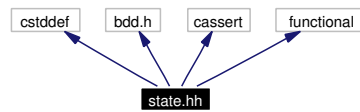
```
#include <cstdint>
```

```
#include <bdd.h>
```

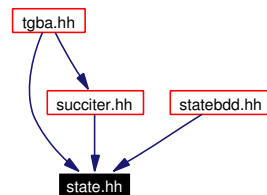
```
#include <cassert>
```

```
#include <functional>
```

Include dependency graph for state.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

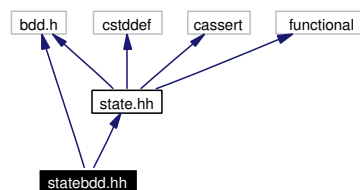
- namespace `spot`

8.42 statebdd.hh File Reference

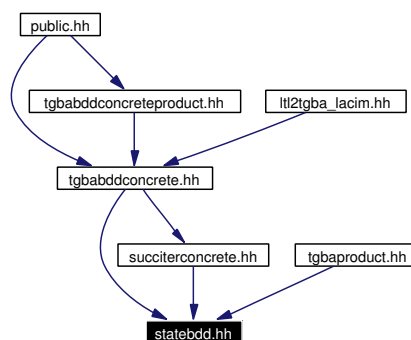
```
#include <bdd.h>
```

```
#include "state.hh"
```

Include dependency graph for `statebdd.hh`:



This graph shows which files directly or indirectly include this file:



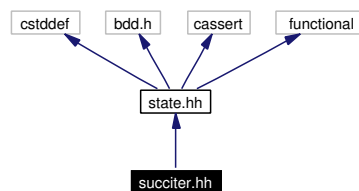
Namespaces

- namespace [spot](#)

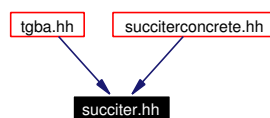
8.43 succiter.hh File Reference

```
#include "state.hh"
```

Include dependency graph for succiter.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [spot](#)

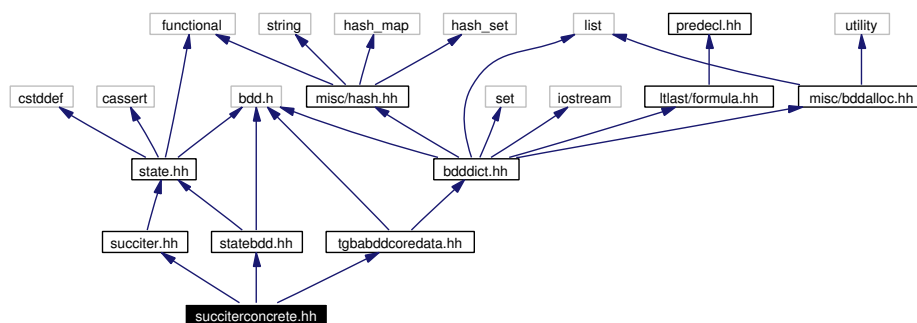
8.44 succiterconcrete.hh File Reference

```
#include "statebdd.hh"
```

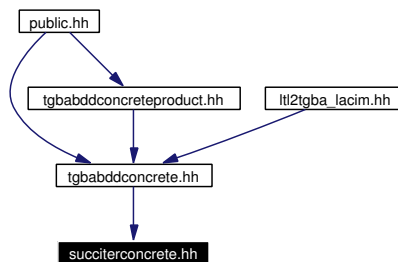
```
#include "succiter.hh"
```

```
#include "tgbabddcoredata.hh"
```

Include dependency graph for succiterconcrete.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace `spot`

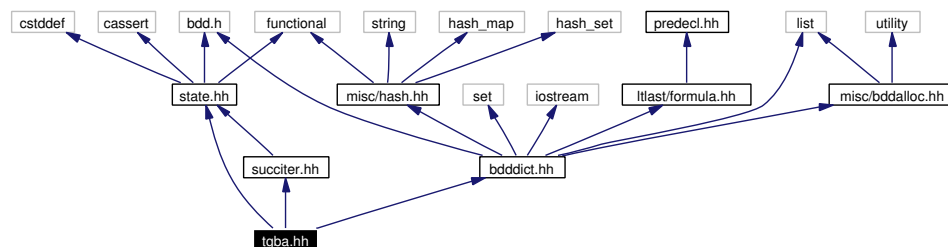
8.45 tgba.hh File Reference

```
#include "state.hh"
```

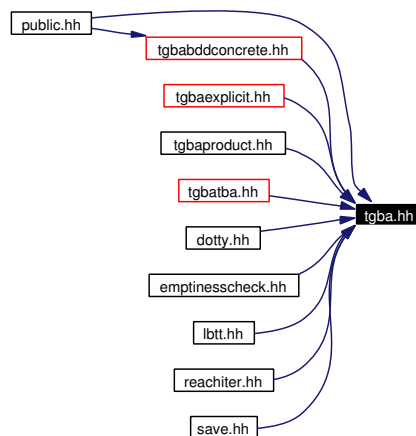
```
#include "succiter.hh"
```

```
#include "bdddict.hh"
```

Include dependency graph for `tgba.hh`:



This graph shows which files directly or indirectly include this file:



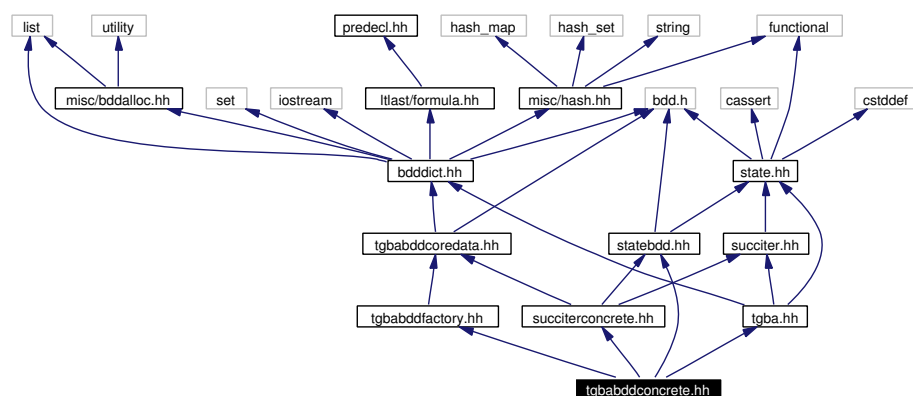
Namespaces

- namespace **spot**

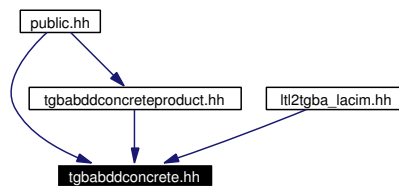
8.46 tgbabddconcrete.hh File Reference

```
#include "tgba.hh"
#include "statebdd.hh"
#include "tgbabddfactory.hh"
#include "succiterconcrete.hh"
```

Include dependency graph for `tgbabddconcrete.hh`:



This graph shows which files directly or indirectly include this file:



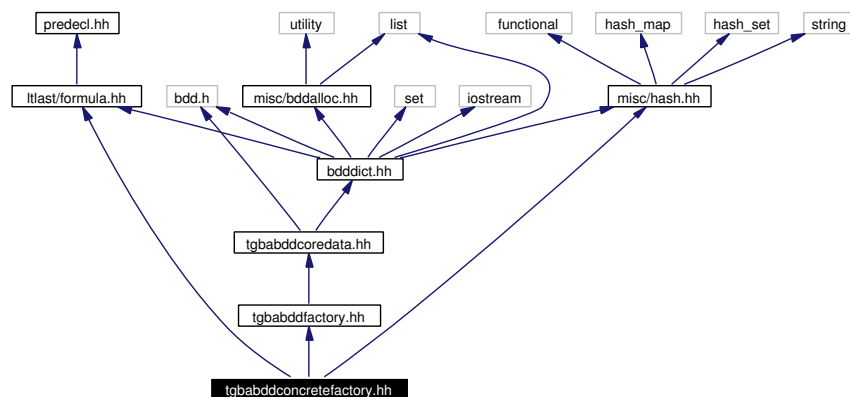
Namespaces

- namespace [spot](#)

8.47 tgbabddconcretefactory.hh File Reference

```
#include "misc/hash.hh"
#include "ltlast/formula.hh"
#include "tgbabddfactory.hh"
```

Include dependency graph for tgbabddconcretefactory.hh:



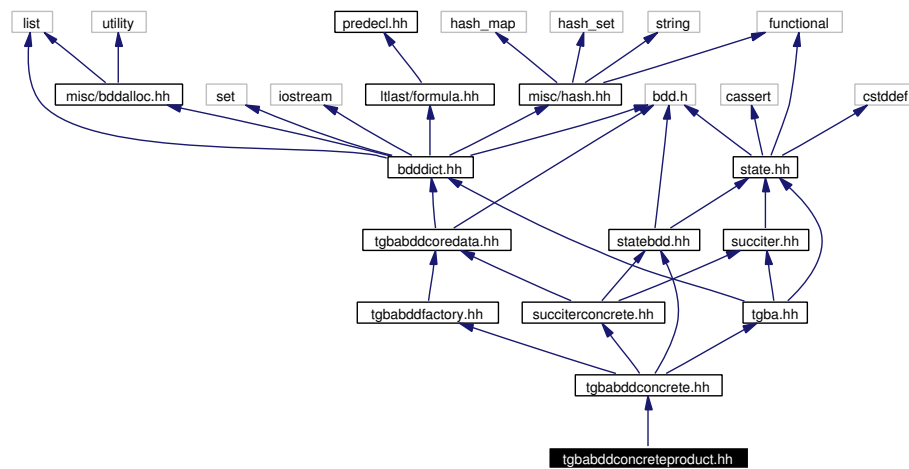
Namespaces

- namespace [spot](#)

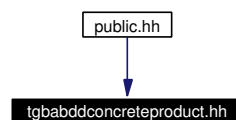
8.48 tgbabddconcreteproduct.hh File Reference

```
#include "tgbabddconcrete.hh"
```

Include dependency graph for tgbabddconcreteproduct.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

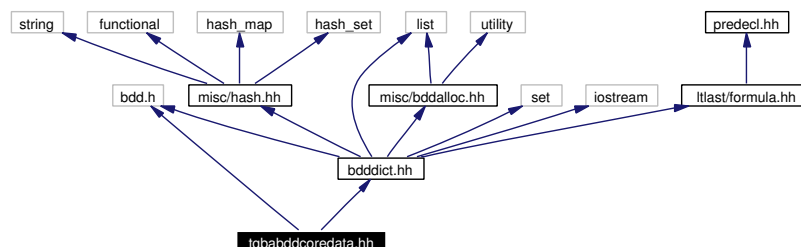
- namespace [spot](#)

8.49 tgbabddcoredata.hh File Reference

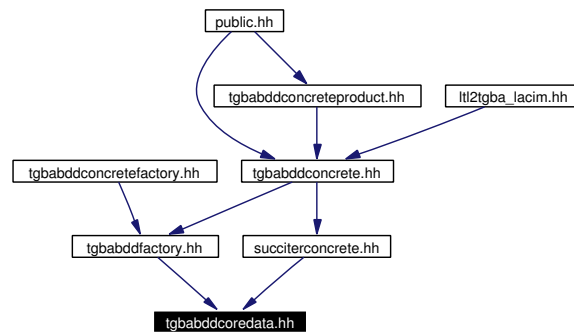
```
#include <bdd.h>
```

```
#include "bdddict.hh"
```

Include dependency graph for tgbabddcoredata.hh:



This graph shows which files directly or indirectly include this file:



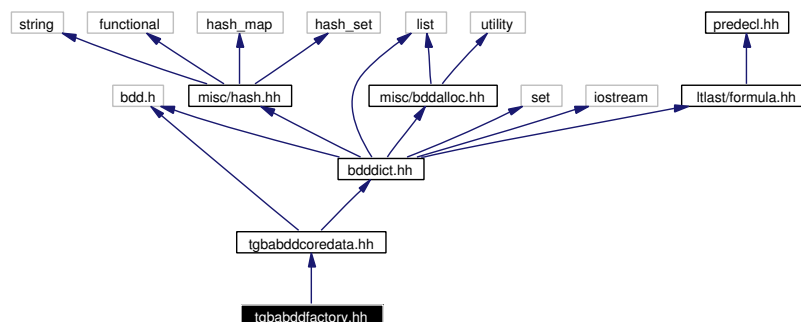
Namespaces

- namespace [spot](#)

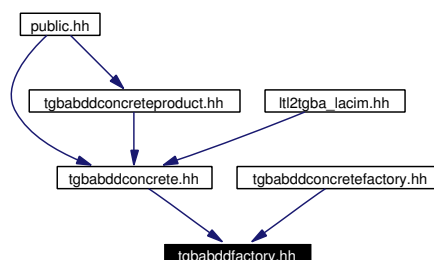
8.50 tgbabddfactory.hh File Reference

```
#include "tgbabddcoredata.hh"
```

Include dependency graph for tgbabddfactory.hh:



This graph shows which files directly or indirectly include this file:



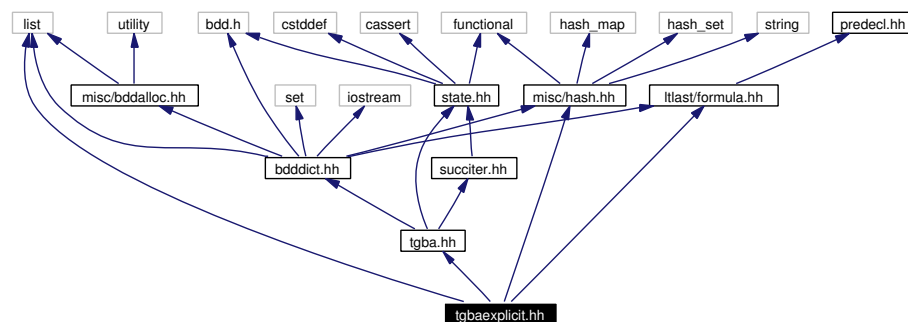
Namespaces

- namespace [spot](#)

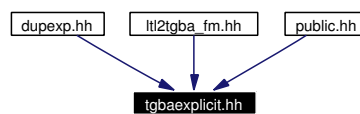
8.51 tgbaexplicit.hh File Reference

```
#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:



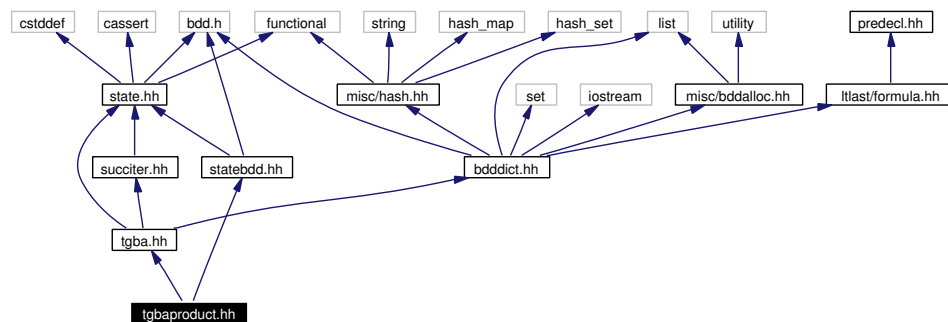
Namespaces

- namespace `spot`

8.52 tgbaproduct.hh File Reference

```
#include "tgba.hh"
#include "statebdd.hh"
```

Include dependency graph for tgbaproduct.hh:



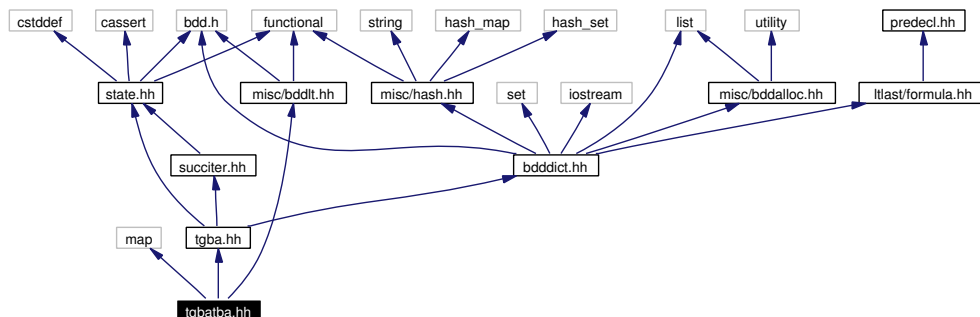
Namespaces

- namespace [spot](#)

8.53 tgbatba.hh File Reference

```
#include <map>
#include "tgba.hh"
#include "misc/bddltt.hh"
```

Include dependency graph for tgbatba.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

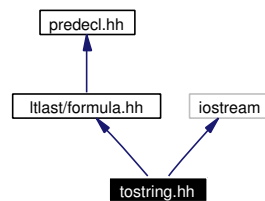
- namespace [spot](#)

8.54 tostring.hh File Reference

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

```
#include <iostream>
```

Include dependency graph for tostring.hh:



Namespaces

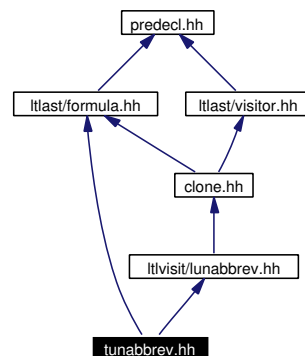
- namespace `spot`
- namespace `spot::ltl`

8.55 tunabbrev.hh File Reference

```
#include "ltlast/formula.hh"
```

```
#include "ltlvisit/ltlabbrev.hh"
```

Include dependency graph for tunabbrev.hh:



Namespaces

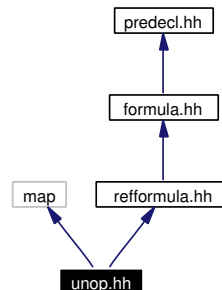
- namespace `spot`
- namespace `spot::ltl`

8.56 unop.hh File Reference

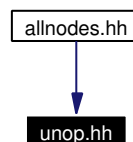
```
#include <map>
```

```
#include "reformula.hh"
```

Include dependency graph for unop.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace `spot`
- namespace `spot::ltl`

8.57 version.hh File Reference

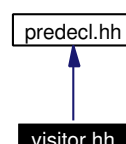
Namespaces

- namespace `spot`

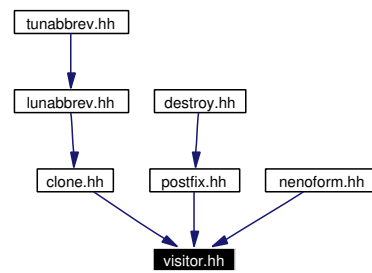
8.58 visitor.hh File Reference

```
#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](#)

Index

- ~atomic_prop
 - spot::ltl::atomic_prop, [24](#)
- ~bdd_dict
 - spot::bdd_dict, [32](#)
- ~binop
 - spot::ltl::binop, [38](#)
- ~clone_visitor
 - spot::ltl::clone_visitor, [41](#)
- ~constant
 - spot::ltl::constant, [45](#)
- ~default_environment
 - spot::ltl::default_environment, [47](#)
- ~emptiness_check
 - spot::emptiness_check, [49](#)
- ~environment
 - spot::ltl::environment, [53](#)
- ~formula
 - spot::ltl::formula, [55](#)
- ~magic_search
 - spot::magic_search, [59](#)
- ~multop
 - spot::ltl::multop, [68](#)
- ~postfix_visitor
 - spot::ltl::postfix_visitor, [73](#)
- ~ref_formula
 - spot::ltl::ref_formula, [75](#)
- ~state
 - spot::state, [80](#)
- ~state_explicit
 - spot::state_explicit, [84](#)
- ~state_product
 - spot::state_product, [86](#)
- ~tgba
 - spot::tgba, [91](#)
- ~tgba_bdd_concrete
 - spot::tgba_bdd_concrete, [96](#)
- ~tgba_bdd_concrete_factory
 - spot::tgba_bdd_concrete_factory, [101](#)
- ~tgba_explicit
 - spot::tgba_explicit, [110](#)
- ~tgba_explicit_succ_iterator
 - spot::tgba_explicit_succ_iterator, [116](#)
- ~tgba_product
 - spot::tgba_product, [119](#)
- ~tgba_reachable_iterator
 - spot::tgba_reachable_iterator, [124](#)
- ~tgba_succ_iterator
 - spot::tgba_succ_iterator, [132](#)
- ~tgba_succ_iterator_concrete
 - spot::tgba_succ_iterator_concrete, [135](#)
- ~tgba_succ_iterator_product
 - spot::tgba_succ_iterator_product, [138](#)
- ~tgba_tba_proxy
 - spot::tgba_tba_proxy, [142](#)
- ~unabbreviate_logic_visitor
 - spot::ltl::unabbreviate_logic_visitor, [146](#)
- ~unabbreviate_ltl_visitor
 - spot::ltl::unabbreviate_ltl_visitor, [149](#)
- ~unop
 - spot::ltl::unop, [152](#)
- a
 - spot::magic_search, [60](#)
- a_
 - spot::tgba_tba_proxy, [145](#)
- acc_
 - spot::tgba_bdd_concrete_factory, [102](#)
- acc_cycle_
 - spot::tgba_tba_proxy, [145](#)
- acc_formula_map
 - spot::bdd_dict, [34](#)
- acc_map
 - spot::bdd_dict, [34](#)
- acc_map_
 - spot::tgba_bdd_concrete_factory, [101](#)
- acc_set
 - spot::tgba_bdd_core_data, [105](#)
- accept
 - spot::ltl::atomic_prop, [25](#)
 - spot::ltl::binop, [38](#)
 - spot::ltl::constant, [45](#)
 - spot::ltl::formula, [55](#)
 - spot::ltl::multop, [68](#)
 - spot::ltl::ref_formula, [76](#)
 - spot::ltl::unop, [152](#)
- acceptance_conditions
 - spot::tgba_bdd_core_data, [105](#)
 - spot::tgba_explicit::transition, [115](#)
- accepting_path
 - spot::emptiness_check, [50](#)
- add_acceptance_condition
 - spot::tgba_explicit, [110](#)
- add_acceptance_conditions
 - spot::tgba_explicit, [110](#)
- add_condition
 - spot::tgba_explicit, [110](#)
- add_conditions
 - spot::tgba_explicit, [111](#)
- add_state
 - spot::tgba_explicit, [111](#)
 - spot::tgba_reachable_iterator, [124](#)

- spot::tgba_reachable_iterator_breadth_first, 127
- spot::tgba_reachable_iterator_depth_first, 130
- all_acceptance_conditions
 - spot::tgba, 91
 - spot::tgba_bdd_concrete, 96
 - spot::tgba_bdd_core_data, 105
 - spot::tgba_explicit, 111
 - spot::tgba_product, 120
 - spot::tgba_tba_proxy, 143
- all_acceptance_conditions_
 - spot::tgba_explicit, 114
 - spot::tgba_explicit_succ_iterator, 117
 - spot::tgba_product, 122
- all_acceptance_conditions_computed_
 - spot::tgba_explicit, 114
- allnodes.hh, 155
- allocate_variables
 - spot::bdd_allocator, 28
 - spot::bdd_dict, 32
- And
 - spot::ltl::multop, 67
- as_bdd
 - spot::state_bdd, 82
- assert_emptyiness
 - spot::bdd_dict, 32
- atomic_prop
 - spot::ltl::atomic_prop, 24
- atomic_prop.hh, 155
- aut_
 - spot::emptiness_check, 50
- automata_
 - spot::tgba_reachable_iterator, 125
 - spot::tgba_reachable_iterator_breadth_first, 128
 - spot::tgba_reachable_iterator_depth_first, 131
- bdd_allocator
 - spot::bdd_allocator, 27
- bdd_dict
 - spot::bdd_dict, 32
- bdd_format_formula
 - spot, 12
- bdd_format_sat
 - spot, 12
- bdd_format_set
 - spot, 12
- bdd_print_acc
 - spot, 12
- bdd_print_accset
 - spot, 13
- bdd_print_dot
 - spot, 13
- bdd_print_formula
 - spot, 13
- bdd_print_sat
 - spot, 13
- bdd_print_set
 - spot, 14
- bdd_print_table
 - spot, 14
- bdd_to_formula
 - spot, 14
- bddalloc.hh, 156
- bdddict.hh, 156
- bddlt.hh, 157
- bddprint.hh, 158
- begin
 - yy::Location, 57
 - yy::Stack, 79
- binop
 - spot::ltl::binop, 38
- binop.hh, 158
- check
 - spot::emptiness_check, 50
 - spot::magic_search, 60
- child
 - spot::ltl::unop, 152
- child_
 - spot::ltl::unop, 153
- children_
 - spot::ltl::multop, 69
- clone
 - spot::ltl, 19
 - spot::state, 80
 - spot::state_bdd, 82
 - spot::state_explicit, 84
 - spot::state_product, 86
- clone.hh, 159
- clone_visitor
 - spot::ltl::clone_visitor, 41
- column
 - yy::Position, 71
- columns
 - yy::Location, 57
 - yy::Position, 71
- compare
 - spot::state, 80
 - spot::state_bdd, 82
 - spot::state_explicit, 84
 - spot::state_product, 86
- complement_all_acceptance_conditions
 - spot::tgba_explicit, 111
- complete_cycle
 - spot::emptiness_check, 50

- compute_support_conditions
 - spot::tgba, 92
 - spot::tgba_bdd_concrete, 97
 - spot::tgba_explicit, 111
 - spot::tgba_product, 120
 - spot::tgba_tba_proxy, 143
- compute_support_variables
 - spot::tgba, 92
 - spot::tgba_bdd_concrete, 97
 - spot::tgba_explicit, 111
 - spot::tgba_product, 120
 - spot::tgba_tba_proxy, 143
- condition
 - spot::emptiness_check::connected_ component, 51
 - spot::emptiness_check::connected_ component_set, 52
 - spot::tgba_explicit::transition, 115
- connected_component
 - spot::emptiness_check::connected_ component, 51
- constant
 - spot::ltl::constant, 45
- constant.hh, 160
- ConstIterator
 - yy::Stack, 78
- constrain_relation
 - spot::tgba_bdd_concrete_factory, 101
- counter_example
 - spot::emptiness_check, 50
- create_atomic_prop
 - spot::tgba_bdd_concrete_factory, 101
- create_state
 - spot::tgba_bdd_concrete_factory, 101
- create_transition
 - spot::tgba_explicit, 111
- cube_
 - spot::minato_isop, 63
- current_
 - spot::tgba_succ_iterator_concrete, 136
- current_acc_
 - spot::tgba_succ_iterator_concrete, 136
- current_acceptance_conditions
 - spot::tgba_explicit_succ_iterator, 116
 - spot::tgba_succ_iterator, 132
 - spot::tgba_succ_iterator_concrete, 135
 - spot::tgba_succ_iterator_product, 139
- current_cond_
 - spot::tgba_succ_iterator_product, 140
- current_condition
 - spot::tgba_explicit_succ_iterator, 116
 - spot::tgba_succ_iterator, 132
 - spot::tgba_succ_iterator_concrete, 135
 - spot::tgba_succ_iterator_product, 139
- current_state
 - spot::tgba_explicit_succ_iterator, 116
 - spot::tgba_succ_iterator, 132
 - spot::tgba_succ_iterator_concrete, 136
 - spot::tgba_succ_iterator_product, 139
- current_state_
 - spot::tgba_succ_iterator_concrete, 137
- cycle_map
 - spot::tgba_tba_proxy, 142
- cycle_path
 - spot::emptiness_check, 49
- data_
 - spot::tgba_bdd_concrete, 99
 - spot::tgba_bdd_concrete_factory, 102
 - spot::tgba_succ_iterator_concrete, 137
- declare_acceptance_condition
 - spot::tgba_bdd_concrete_factory, 101
 - spot::tgba_bdd_core_data, 105
 - spot::tgba_explicit, 111
- declare_atomic_prop
 - spot::tgba_bdd_core_data, 105
- declare_now_next
 - spot::tgba_bdd_core_data, 105
- default_environment
 - spot::ltl::default_environment, 47
- defaultenv.hh, 160
- dest
 - spot::tgba_explicit::transition, 115
- destroy
 - spot::ltl, 19
- destroy.hh, 161
- dict
 - spot::tgba_bdd_core_data, 106
- dict_
 - spot::tgba_explicit, 114
 - spot::tgba_product, 122
- doit
 - spot::ltl::postfix_visitor, 73
- doit_default
 - spot::ltl::postfix_visitor, 73
- done
 - spot::tgba_explicit_succ_iterator, 116
 - spot::tgba_succ_iterator, 133
 - spot::tgba_succ_iterator_concrete, 136
 - spot::tgba_succ_iterator_product, 139
- dotty
 - spot::ltl, 19
- dotty.hh, 161, 162
- dotty_reachable
 - spot, 14
- dump
 - spot::bdd_dict, 32
 - spot::ltl, 19

- dump.hh, 162
- dupexp.hh, 163
- emptiness_check
 - spot::emptiness_check, 49
- emptinesscheck.hh, 163
- end
 - spot::tgba_reachable_iterator, 124
 - spot::tgba_reachable_iterator_breadth_first, 127
 - spot::tgba_reachable_iterator_depth_first, 130
 - yy::Location, 57
 - yy::Stack, 79
- env
 - spot::ltl::atomic_prop, 25
- env_
 - spot::ltl::atomic_prop, 26
- environment.hh, 164
- Equiv
 - spot::ltl::binop, 38
- extvarnum
 - spot::bdd_allocator, 28
- F
 - spot::ltl::unop, 152
- f0_max
 - spot::minato_isop::local_vars, 64
- f0_min
 - spot::minato_isop::local_vars, 64
- f1_max
 - spot::minato_isop::local_vars, 64
- f1_min
 - spot::minato_isop::local_vars, 64
- f_max
 - spot::minato_isop::local_vars, 64
- f_min
 - spot::minato_isop::local_vars, 65
- False
 - spot::ltl::constant, 44
- false_instance
 - spot::ltl::constant, 45
- filename
 - yy::Position, 71
- finish
 - spot::tgba_bdd_concrete_factory, 102
- first
 - spot::ltl::binop, 39
 - spot::tgba_explicit_succ_iterator, 117
 - spot::tgba_succ_iterator, 133
 - spot::tgba_succ_iterator_concrete, 136
 - spot::tgba_succ_iterator_product, 139
- first_
 - spot::ltl::binop, 40
- FirstStep
 - spot::minato_isop::local_vars, 64
- format_parse_errors
 - spot::ltl, 19
- format_state
 - spot::tgba, 92
 - spot::tgba_bdd_concrete, 97
 - spot::tgba_explicit, 111, 112
 - spot::tgba_product, 120
 - spot::tgba_tba_proxy, 143
- format_tgba_parse_errors
 - spot, 14
- formula.hh, 164
- formula2bdd.hh, 165
- formula_to_bdd
 - spot, 14
- FourthStep
 - spot::minato_isop::local_vars, 64
- free_list
 - spot::bdd_allocator, 28
 - spot::bdd_dict, 34
- free_list_type
 - spot::bdd_allocator, 27
 - spot::bdd_dict, 31
- fv_map
 - spot::bdd_dict, 31
- G
 - spot::ltl::unop, 152
- g0
 - spot::minato_isop::local_vars, 65
- g1
 - spot::minato_isop::local_vars, 65
- get_acceptance_condition
 - spot::tgba_explicit, 112
- get_core_data
 - spot::tgba_bdd_concrete, 97
 - spot::tgba_bdd_concrete_factory, 102
 - spot::tgba_bdd_factory, 107
- get_dict
 - spot::tgba, 92
 - spot::tgba_bdd_concrete, 97
 - spot::tgba_bdd_concrete_factory, 102
 - spot::tgba_explicit, 112
 - spot::tgba_product, 120
 - spot::tgba_tba_proxy, 143
- get_init_bdd
 - spot::tgba_bdd_concrete, 97
- get_init_state
 - spot::tgba, 92
 - spot::tgba_bdd_concrete, 97
 - spot::tgba_explicit, 112
 - spot::tgba_product, 120
 - spot::tgba_tba_proxy, 143

- get_state
 - spot::state_explicit, 84
- h
 - spot::emptiness_check, 50
 - spot::magic_search, 60
- h_filt
 - spot::emptiness_check, 50
- has
 - spot::magic_search, 60
- has_acceptance_condition
 - spot::tgba_explicit, 112
- has_state
 - spot::emptiness_check::connected_ - component_set, 52
- hash
 - spot::state, 80
 - spot::state_bdd, 82
 - spot::state_explicit, 84
 - spot::state_product, 87
- hash.hh, 166
- hash_type
 - spot::emptiness_check, 49
 - spot::magic_search, 59
- height
 - yy::Stack, 79
- i_
 - spot::tgba_explicit_succ_iterator, 117
- Implies
 - spot::ltl::binop, 38
- index
 - spot::emptiness_check::connected_ - component, 51
 - spot::emptiness_check::connected_ - component_set, 52
- init_
 - spot::tgba_bdd_concrete, 99
 - spot::tgba_explicit, 114
- initial_column
 - yy::Position, 71
- initial_line
 - yy::Position, 72
- initialize
 - spot::bdd_allocator, 28
 - spot::bdd_dict, 32
- initialized
 - spot::bdd_allocator, 28
 - spot::bdd_dict, 34
- instance
 - spot::ltl::atomic_prop, 25
 - spot::ltl::binop, 39
 - spot::ltl::default_environment, 47
 - spot::ltl::multop, 68
 - spot::ltl::unop, 152
- instance_count
 - spot::ltl::atomic_prop, 25
 - spot::ltl::binop, 39
 - spot::ltl::multop, 68
 - spot::ltl::unop, 152
- instances
 - spot::ltl::atomic_prop, 26
 - spot::ltl::binop, 40
 - spot::ltl::multop, 69
 - spot::ltl::unop, 153
- is_registered_acceptance_variable
 - spot::bdd_dict, 32
- is_registered_proposition
 - spot::bdd_dict, 32
- is_registered_state
 - spot::bdd_dict, 32
- Iterator
 - yy::Stack, 78
- last_support_conditions_input_
 - spot::tgba, 94
- last_support_conditions_output_
 - spot::tgba, 94
- last_support_variables_input_
 - spot::tgba, 94
- last_support_variables_output_
 - spot::tgba, 94
- lbtt.hh, 166
- lbtt_reachable
 - spot, 14
- left
 - spot::state_product, 87
- left_
 - spot::state_product, 87
 - spot::tgba_product, 122
 - spot::tgba_succ_iterator_product, 140
- left_acc_complement_
 - spot::tgba_product, 122
- left_neg_
 - spot::tgba_succ_iterator_product, 140
- line
 - yy::Position, 72
- lines
 - yy::Location, 57
 - yy::Position, 71
- local_vars
 - spot::minato_isop::local_vars, 64
- Location
 - yy::Location, 57
- location.hh, 167
- ltl2tgba_fm.hh, 168
- ltl2tgba_lacim.hh, 168
- ltl_to_tgba_fm

- spot, 15
- ltl_to_tgba_lacim
 - spot, 15
- lunabbrev.hh, 169
- lvarnum
 - spot::bdd_allocator, 28
 - spot::bdd_dict, 34
- m
 - spot::magic_search::magic_state, 62
- magic.hh, 170
- magic_search
 - spot::magic_search, 59
- mainpage.dox, 170
- map
 - spot::ltl::atomic_prop, 24
 - spot::ltl::binop, 38
 - spot::ltl::multop, 67
 - spot::ltl::unop, 152
- minato.hh, 170
- minato_isop
 - spot::minato_isop, 63
- multop
 - spot::ltl::multop, 68
- multop.hh, 171
- name
 - spot::ltl::atomic_prop, 25
 - spot::ltl::default_environment, 47
 - spot::ltl::environment, 53
- name_
 - spot::ltl::atomic_prop, 26
- name_state_map_
 - spot::tgba_explicit, 114
- neg_acceptance_conditions
 - spot::tgba, 92
 - spot::tgba_bdd_concrete, 98
 - spot::tgba_explicit, 112
 - spot::tgba_product, 120
 - spot::tgba_tba_proxy, 143
- neg_acceptance_conditions_
 - spot::tgba_explicit, 114
 - spot::tgba_product, 122
- negacc_set
 - spot::tgba_bdd_core_data, 106
- negative_normal_form
 - spot::ltl, 19
- nenofrm.hh, 171
- next
 - spot::minato_isop, 63
 - spot::tgba_explicit_succ_iterator, 117
 - spot::tgba_succ_iterator, 133
 - spot::tgba_succ_iterator_concrete, 136
 - spot::tgba_succ_iterator_product, 139
- next_non_false_
 - spot::tgba_succ_iterator_product, 140
- next_set
 - spot::tgba_bdd_core_data, 106
- next_state
 - spot::tgba_reachable_iterator, 124
 - spot::tgba_reachable_iterator_breadth_first, 127
 - spot::tgba_reachable_iterator_depth_first, 130
- next_to_now
 - spot::bdd_dict, 34
- Not
 - spot::ltl::unop, 152
- notacc_set
 - spot::tgba_bdd_core_data, 106
- notnext_set
 - spot::tgba_bdd_core_data, 106
- notnow_set
 - spot::tgba_bdd_core_data, 106
- notvar_set
 - spot::tgba_bdd_core_data, 106
- now_formula_map
 - spot::bdd_dict, 34
- now_map
 - spot::bdd_dict, 34
- now_set
 - spot::tgba_bdd_core_data, 106
- now_to_next
 - spot::bdd_dict, 35
- nownext_set
 - spot::tgba_bdd_core_data, 106
- ns_map
 - spot::tgba_explicit, 110
- nth
 - spot::ltl::multop, 68
- op
 - spot::ltl::binop, 39
 - spot::ltl::multop, 69
 - spot::ltl::unop, 152
- op_
 - spot::ltl::binop, 40
 - spot::ltl::multop, 69
 - spot::ltl::unop, 153
- op_name
 - spot::ltl::binop, 39
 - spot::ltl::multop, 69
 - spot::ltl::unop, 153
- operator()
 - spot::bdd_less_than, 35
 - spot::ltl::multop::paircmp, 70
 - spot::ptr_hash, 74
 - spot::state_ptr_equal, 88

- spot::state_ptr_hash, 88
- spot::state_ptr_less_than, 89
- spot::string_hash, 89
- operator+
 - yy, 22
- operator+=
 - yy, 22
- operator-
 - yy, 22
- operator-=
 - yy, 22
- operator<<
 - yy, 22
- operator=
 - spot::bdd_dict, 33
 - spot::tgba_bdd_core_data, 105
- operator[]
 - yy::Slice, 77
 - yy::Stack, 79
- Or
 - spot::ltl::multop, 67
- pair
 - spot::ltl::atomic_prop, 24
 - spot::ltl::binop, 38
 - spot::ltl::multop, 67
 - spot::ltl::unop, 152
- pairf
 - spot::ltl::binop, 38
- parse
 - spot::ltl, 20
- parse_error
 - spot::ltl, 18
- parse_error_list
 - spot::ltl, 18
- period
 - spot::emptiness_check, 50
- pop
 - yy::Stack, 79
- pos_lenght_pair
 - spot::bdd_allocator, 27
 - spot::bdd_dict, 31
- Position
 - yy::Position, 71
- position.hh, 172
- postfix.hh, 172
- postfix_visitor
 - spot::ltl::postfix_visitor, 73
- predecl.hh, 173
- print_result
 - spot::emptiness_check, 50
 - spot::magic_search, 60
- process_link
 - spot::tgba_reachable_iterator, 124
- spot::tgba_reachable_iterator_breadth_first, 127
- spot::tgba_reachable_iterator_depth_first, 130
- process_state
 - spot::tgba_reachable_iterator, 125
 - spot::tgba_reachable_iterator_breadth_first, 127
 - spot::tgba_reachable_iterator_depth_first, 130
- product
 - spot, 15
- project_state
 - spot::tgba, 93
 - spot::tgba_bdd_concrete, 98
 - spot::tgba_explicit, 112
 - spot::tgba_product, 121
 - spot::tgba_tba_proxy, 144
- public.hh, 173–175
- push
 - spot::magic_search, 60
 - yy::Stack, 79
- R
 - spot::ltl::binop, 38
- range_
 - yy::Slice, 77
- reachiter.hh, 175
- recurse
 - spot::ltl::clone_visitor, 41
 - spot::ltl::unabbreviate_logic_visitor, 147
 - spot::ltl::unabbreviate_ltl_visitor, 149
- ref
 - spot::ltl::atomic_prop, 25
 - spot::ltl::binop, 39
 - spot::ltl::constant, 45
 - spot::ltl::formula, 55
 - spot::ltl::multop, 69
 - spot::ltl::ref_formula, 76
 - spot::ltl::unop, 153
- ref_
 - spot::ltl::atomic_prop, 25
 - spot::ltl::binop, 39
 - spot::ltl::constant, 45
 - spot::ltl::formula, 55
 - spot::ltl::multop, 69
 - spot::ltl::ref_formula, 76
 - spot::ltl::unop, 153
- ref_count_
 - spot::ltl::ref_formula, 76
- ref_formula
 - spot::ltl::ref_formula, 75
- ref_set
 - spot::bdd_dict, 31

- reformula.hh, 176
- register_acceptance_variable
 - spot::bdd_dict, 33
- register_acceptance_variables
 - spot::bdd_dict, 33
- register_all_variables_of
 - spot::bdd_dict, 33
- register_proposition
 - spot::bdd_dict, 33
- register_propositions
 - spot::bdd_dict, 33
- register_state
 - spot::bdd_dict, 33
- relation
 - spot::tgba_bdd_core_data, 106
- release_variables
 - spot::bdd_allocator, 28
 - spot::bdd_dict, 34
- remove_component
 - spot::emptiness_check, 50
- require
 - spot::ltl::default_environment, 47
 - spot::ltl::environment, 53
- result
 - spot::ltl::clone_visitor, 41
 - spot::ltl::unabbreviate_logic_visitor, 147
 - spot::ltl::unabbreviate_ltl_visitor, 149
- result_
 - spot::ltl::clone_visitor, 42
 - spot::ltl::unabbreviate_logic_visitor, 147
 - spot::ltl::unabbreviate_ltl_visitor, 149
- ret_
 - spot::minato_isop, 63
- right
 - spot::state_product, 87
- right_
 - spot::state_product, 87
 - spot::tgba_product, 122
 - spot::tgba_succ_iterator_product, 140
- right_acc_complement_
 - spot::tgba_product, 122
- right_neg_
 - spot::tgba_succ_iterator_product, 140
- root
 - spot::emptiness_check, 50
- run
 - spot::tgba_reachable_iterator, 125
 - spot::tgba_reachable_iterator_breadth_first, 127
 - spot::tgba_reachable_iterator_depth_first, 130
- s
 - spot::magic_search::magic_state, 62
- s_
 - spot::tgba_explicit_succ_iterator, 117
- save.hh, 176
- second
 - spot::ltl::binop, 39
- second_
 - spot::ltl::binop, 40
- SecondStep
 - spot::minato_isop::local_vars, 64
- seen
 - spot::tgba_reachable_iterator, 125
 - spot::tgba_reachable_iterator_breadth_first, 128
 - spot::tgba_reachable_iterator_depth_first, 131
- seen_map
 - spot::tgba_reachable_iterator, 124
 - spot::tgba_reachable_iterator_breadth_first, 127
 - spot::tgba_reachable_iterator_depth_first, 130
- seen_with
 - spot::magic_search::magic, 61
- seen_without
 - spot::magic_search::magic, 61
- seq_
 - yy::Stack, 79
- set_init_state
 - spot::tgba_bdd_concrete, 98
 - spot::tgba_explicit, 112
- set_type
 - spot::emptiness_check::connected_component_set, 52
- size
 - spot::ltl::multop, 69
- Slice
 - yy::Slice, 77
- sn_map
 - spot::tgba_explicit, 110
- spot, 8
 - bdd_format_formula, 12
 - bdd_format_sat, 12
 - bdd_format_set, 12
 - bdd_print_acc, 12
 - bdd_print_accset, 13
 - bdd_print_dot, 13
 - bdd_print_formula, 13
 - bdd_print_sat, 13
 - bdd_print_set, 14
 - bdd_print_table, 14
 - bdd_to_formula, 14
 - dotty_reachable, 14
 - format_tgba_parse_errors, 14
 - formula_to_bdd, 14

- lbt_reachable, 14
- ltl_to_tgba_fm, 15
- ltl_to_tgba_lacim, 15
- product, 15
- tgba_dupexp_bfs, 16
- tgba_dupexp_dfs, 16
- tgba_parse, 16
- tgba_parse_error, 12
- tgba_parse_error_list, 12
- tgba_save_reachable, 16
- version, 16
- spot::bdd_allocator, 26
 - allocate_variables, 28
 - bdd_allocator, 27
 - extvarnum, 28
 - free_list, 28
 - free_list_type, 27
 - initialize, 28
 - initialized, 28
 - lvarnum, 28
 - pos_lenght_pair, 27
 - release_variables, 28
 - varnum, 28
- spot::bdd_dict, 28
 - ~bdd_dict, 32
 - acc_formula_map, 34
 - acc_map, 34
 - allocate_variables, 32
 - assert_emptiness, 32
 - bdd_dict, 32
 - dump, 32
 - free_list, 34
 - free_list_type, 31
 - fv_map, 31
 - initialize, 32
 - initialized, 34
 - is_registered_acceptance_variable, 32
 - is_registered_proposition, 32
 - is_registered_state, 32
 - lvarnum, 34
 - next_to_now, 34
 - now_formula_map, 34
 - now_map, 34
 - now_to_next, 35
 - operator=, 33
 - pos_lenght_pair, 31
 - ref_set, 31
 - register_acceptance_variable, 33
 - register_acceptance_variables, 33
 - register_all_variables_of, 33
 - register_proposition, 33
 - register_propositions, 33
 - register_state, 33
 - release_variables, 34
 - unregister_all_my_variables, 34
 - var_formula_map, 35
 - var_map, 35
 - var_refs, 35
 - varnum, 35
 - vf_map, 32
 - vr_map, 32
- spot::bdd_less_than, 35
 - operator(), 35
- spot::emptiness_check, 48
 - ~emptiness_check, 49
 - accepting_path, 50
 - aut_, 50
 - check, 50
 - complete_cycle, 50
 - counter_example, 50
 - cycle_path, 49
 - emptiness_check, 49
 - h, 50
 - h_filt, 50
 - hash_type, 49
 - period, 50
 - print_result, 50
 - remove_component, 50
 - root, 50
 - state_proposition, 49
 - state_sequence, 49
 - suffix, 50
- spot::emptiness_check::connected_component, 51
 - condition, 51
 - connected_component, 51
 - index, 51
- spot::emptiness_check::connected_component_set, 51
 - condition, 52
 - has_state, 52
 - index, 52
 - set_type, 52
 - states, 52
- spot::ltl, 16
 - clone, 19
 - destroy, 19
 - dotty, 19
 - dump, 19
 - format_parse_errors, 19
 - negative_normal_form, 19
 - parse, 20
 - parse_error, 18
 - parse_error_list, 18
 - to_string, 20
 - unabbreviate_logic, 20
 - unabbreviate_ltl, 21
- spot::ltl::atomic_prop, 23

- ~atomic_prop, 24
- accept, 25
- atomic_prop, 24
- env, 25
- env_, 26
- instance, 25
- instance_count, 25
- instances, 26
- map, 24
- name, 25
- name_, 26
- pair, 24
- ref, 25
- ref_, 25
- unref, 25
- unref_, 25
- spot::ltl::binop, 36
 - ~binop, 38
 - accept, 38
 - binop, 38
 - Equiv, 38
 - first, 39
 - first_, 40
 - Implies, 38
 - instance, 39
 - instance_count, 39
 - instances, 40
 - map, 38
 - op, 39
 - op_, 40
 - op_name, 39
 - pair, 38
 - pairf, 38
 - R, 38
 - ref, 39
 - ref_, 39
 - second, 39
 - second_, 40
 - type, 38
 - U, 38
 - unref, 39
 - unref_, 40
 - Xor, 38
- spot::ltl::clone_visitor, 40
 - ~clone_visitor, 41
 - clone_visitor, 41
 - recurse, 41
 - result, 41
 - result_, 42
 - visit, 41, 42
- spot::ltl::const_visitor, 42
 - visit, 42, 43
- spot::ltl::constant, 43
 - ~constant, 45
 - accept, 45
 - constant, 45
 - False, 44
 - false_instance, 45
 - ref, 45
 - ref_, 45
 - True, 44
 - true_instance, 45
 - type, 44
 - unref, 45
 - unref_, 45
 - val, 46
 - val_, 46
 - val_name, 46
- spot::ltl::default_environment, 46
 - ~default_environment, 47
 - default_environment, 47
 - instance, 47
 - name, 47
 - require, 47
- spot::ltl::environment, 53
 - ~environment, 53
 - name, 53
 - require, 53
- spot::ltl::formula, 54
 - ~formula, 55
 - accept, 55
 - ref, 55
 - ref_, 55
 - unref, 55
 - unref_, 55
- spot::ltl::multop, 65
 - ~multop, 68
 - accept, 68
 - And, 67
 - children_, 69
 - instance, 68
 - instance_count, 68
 - instances, 69
 - map, 67
 - multop, 68
 - nth, 68
 - op, 69
 - op_, 69
 - op_name, 69
 - Or, 67
 - pair, 67
 - ref, 69
 - ref_, 69
 - size, 69
 - type, 67
 - unref, 69
 - unref_, 69
 - vec, 67

- spot::ltl::multop::paircmp, 70
 - operator(), 70
- spot::ltl::postfix_visitor, 72
 - ~postfix_visitor, 73
 - doit, 73
 - doit_default, 73
 - postfix_visitor, 73
 - visit, 73
- spot::ltl::ref_formula, 74
 - ~ref_formula, 75
 - accept, 76
 - ref, 76
 - ref_, 76
 - ref_count_, 76
 - ref_formula, 75
 - unref, 76
 - unref_, 76
- spot::ltl::unabbreviate_logic_visitor, 145
 - ~unabbreviate_logic_visitor, 146
 - recurse, 147
 - result, 147
 - result_, 147
 - super, 146
 - unabbreviate_logic_visitor, 146
 - visit, 147
- spot::ltl::unabbreviate_ltl_visitor, 147
 - ~unabbreviate_ltl_visitor, 149
 - recurse, 149
 - result, 149
 - result_, 149
 - super, 149
 - unabbreviate_ltl_visitor, 149
 - visit, 149
- spot::ltl::unop, 150
 - ~unop, 152
 - accept, 152
 - child, 152
 - child_, 153
 - F, 152
 - G, 152
 - instance, 152
 - instance_count, 152
 - instances, 153
 - map, 152
 - Not, 152
 - op, 152
 - op_, 153
 - op_name, 153
 - pair, 152
 - ref, 153
 - ref_, 153
 - type, 152
 - unop, 152
 - unref, 153
 - unref_, 153
 - X, 152
- spot::ltl::visitor, 153
 - visit, 154
- spot::magic_search, 57
 - ~magic_search, 59
 - a, 60
 - check, 60
 - h, 60
 - has, 60
 - hash_type, 59
 - magic_search, 59
 - print_result, 60
 - push, 60
 - stack, 60
 - stack_type, 59
 - state_iter_pair, 59
 - tstack, 60
 - tstack_type, 59
 - x, 60
- spot::magic_search::magic, 61
 - seen_with, 61
 - seen_without, 61
- spot::magic_search::magic_state, 61
 - m, 62
 - s, 62
- spot::minato_isop, 62
 - cube_, 63
 - minato_isop, 63
 - next, 63
 - ret_, 63
 - todo_, 63
- spot::minato_isop::local_vars
 - FirstStep, 64
 - FourthStep, 64
 - SecondStep, 64
 - ThirdStep, 64
- spot::minato_isop::local_vars, 63
 - f0_max, 64
 - f0_min, 64
 - f1_max, 64
 - f1_min, 64
 - f_max, 64
 - f_min, 65
 - g0, 65
 - g1, 65
 - local_vars, 64
 - step, 65
 - v1, 65
 - vars, 65
- spot::ptr_hash, 74
 - operator(), 74
- spot::state, 79
 - ~state, 80

- clone, 80
- compare, 80
- hash, 80
- spot::state_bdd, 81
 - as_bdd, 82
 - clone, 82
 - compare, 82
 - hash, 82
 - state_, 83
 - state_bdd, 82
- spot::state_explicit, 83
 - ~state_explicit, 84
 - clone, 84
 - compare, 84
 - get_state, 84
 - hash, 84
 - state_, 85
 - state_explicit, 84
- spot::state_product, 85
 - ~state_product, 86
 - clone, 86
 - compare, 86
 - hash, 87
 - left, 87
 - left_, 87
 - right, 87
 - right_, 87
 - state_product, 86
- spot::state_ptr_equal, 87
 - operator(), 88
- spot::state_ptr_hash, 88
 - operator(), 88
- spot::state_ptr_less_than, 89
 - operator(), 89
- spot::string_hash, 89
 - operator(), 89
- spot::tgba, 90
 - ~tgba, 91
 - all_acceptance_conditions, 91
 - compute_support_conditions, 92
 - compute_support_variables, 92
 - format_state, 92
 - get_dict, 92
 - get_init_state, 92
 - last_support_conditions_input_, 94
 - last_support_conditions_output_, 94
 - last_support_variables_input_, 94
 - last_support_variables_output_, 94
 - neg_acceptance_conditions, 92
 - project_state, 93
 - succ_iter, 93
 - support_conditions, 93
 - support_variables, 93
 - tgba, 91
 - spot::tgba_bdd_concrete, 94
 - ~tgba_bdd_concrete, 96
 - all_acceptance_conditions, 96
 - compute_support_conditions, 97
 - compute_support_variables, 97
 - data_, 99
 - format_state, 97
 - get_core_data, 97
 - get_dict, 97
 - get_init_bdd, 97
 - get_init_state, 97
 - init_, 99
 - neg_acceptance_conditions, 98
 - project_state, 98
 - set_init_state, 98
 - succ_iter, 98
 - support_conditions, 99
 - support_variables, 99
 - tgba_bdd_concrete, 96
 - tgba_bdd_concrete::operator=, 99
 - spot::tgba_bdd_concrete_factory, 99
 - ~tgba_bdd_concrete_factory, 101
 - acc_, 102
 - acc_map_, 101
 - constrain_relation, 101
 - create_atomic_prop, 101
 - create_state, 101
 - data_, 102
 - declare_acceptance_condition, 101
 - finish, 102
 - get_core_data, 102
 - get_dict, 102
 - tgba_bdd_concrete_factory, 101
 - spot::tgba_bdd_core_data, 102
 - acc_set, 105
 - acceptance_conditions, 105
 - all_acceptance_conditions, 105
 - declare_acceptance_condition, 105
 - declare_atomic_prop, 105
 - declare_now_next, 105
 - dict, 106
 - negacc_set, 106
 - next_set, 106
 - notacc_set, 106
 - notnext_set, 106
 - notnow_set, 106
 - notvar_set, 106
 - now_set, 106
 - nownext_set, 106
 - operator=, 105
 - relation, 106
 - tgba_bdd_core_data, 104
 - var_set, 106
 - varandnext_set, 107

- spot::tgba_bdd_factory, 107
 - get_core_data, 107
- spot::tgba_explicit, 108
 - ~tgba_explicit, 110
 - add_acceptance_condition, 110
 - add_acceptance_conditions, 110
 - add_condition, 110
 - add_conditions, 111
 - add_state, 111
 - all_acceptance_conditions, 111
 - all_acceptance_conditions_, 114
 - all_acceptance_conditions_computed_, 114
 - complement_all_acceptance_conditions, 111
 - compute_support_conditions, 111
 - compute_support_variables, 111
 - create_transition, 111
 - declare_acceptance_condition, 111
 - dict_, 114
 - format_state, 111, 112
 - get_acceptance_condition, 112
 - get_dict, 112
 - get_init_state, 112
 - has_acceptance_condition, 112
 - init_, 114
 - name_state_map_, 114
 - neg_acceptance_conditions, 112
 - neg_acceptance_conditions_, 114
 - ns_map, 110
 - project_state, 112
 - set_init_state, 112
 - sn_map, 110
 - state, 110
 - state_name_map_, 114
 - succ_iter, 113
 - support_conditions, 113
 - support_variables, 113
 - tgba_explicit, 110
 - tgba_explicit::operator=, 113
- spot::tgba_explicit::transition, 114
 - acceptance_conditions, 115
 - condition, 115
 - dest, 115
- spot::tgba_explicit_succ_iterator, 115
 - ~tgba_explicit_succ_iterator, 116
 - all_acceptance_conditions_, 117
 - current_acceptance_conditions, 116
 - current_condition, 116
 - current_state, 116
 - done, 116
 - first, 117
 - i_, 117
 - next, 117
 - s_, 117
 - tgba_explicit_succ_iterator, 116
- spot::tgba_product, 117
 - ~tgba_product, 119
 - all_acceptance_conditions, 120
 - all_acceptance_conditions_, 122
 - compute_support_conditions, 120
 - compute_support_variables, 120
 - dict_, 122
 - format_state, 120
 - get_dict, 120
 - get_init_state, 120
 - left_, 122
 - left_acc_complement_, 122
 - neg_acceptance_conditions, 120
 - neg_acceptance_conditions_, 122
 - project_state, 121
 - right_, 122
 - right_acc_complement_, 122
 - succ_iter, 121
 - support_conditions, 121
 - support_variables, 122
 - tgba_product, 119
 - tgba_product::operator=, 122
- spot::tgba_reachable_iterator, 122
 - ~tgba_reachable_iterator, 124
 - add_state, 124
 - automata_, 125
 - end, 124
 - next_state, 124
 - process_link, 124
 - process_state, 125
 - run, 125
 - seen, 125
 - seen_map, 124
 - start, 125
 - tgba_reachable_iterator, 124
- spot::tgba_reachable_iterator_breadth_first, 125
 - add_state, 127
 - automata_, 128
 - end, 127
 - next_state, 127
 - process_link, 127
 - process_state, 127
 - run, 127
 - seen, 128
 - seen_map, 127
 - start, 128
 - tgba_reachable_iterator_breadth_first, 127
 - todo, 128
- spot::tgba_reachable_iterator_depth_first, 128
 - add_state, 130
 - automata_, 131
 - end, 130
 - next_state, 130

- process_link, 130
- process_state, 130
- run, 130
- seen, 131
- seen_map, 130
- start, 131
- tgba_reachable_iterator_depth_first, 130
- todo, 131
- spot::tgba_succ_iterator, 131
 - ~tgba_succ_iterator, 132
 - current_acceptance_conditions, 132
 - current_condition, 132
 - current_state, 132
 - done, 133
 - first, 133
 - next, 133
- spot::tgba_succ_iterator_concrete, 133
 - ~tgba_succ_iterator_concrete, 135
 - current_, 136
 - current_acc_, 136
 - current_acceptance_conditions, 135
 - current_condition, 135
 - current_state, 136
 - current_state_, 137
 - data_, 137
 - done, 136
 - first, 136
 - next, 136
 - succ_set_, 137
 - succ_set_left_, 137
 - tgba_succ_iterator_concrete, 135
- spot::tgba_succ_iterator_product, 137
 - ~tgba_succ_iterator_product, 138
 - current_acceptance_conditions, 139
 - current_cond_, 140
 - current_condition, 139
 - current_state, 139
 - done, 139
 - first, 139
 - left_, 140
 - left_neg_, 140
 - next, 139
 - next_non_false_, 140
 - right_, 140
 - right_neg_, 140
 - step_, 140
 - tgba_succ_iterator_product, 138
- spot::tgba_tba_proxy, 140
 - ~tgba_tba_proxy, 142
 - a_, 145
 - acc_cycle_, 145
 - all_acceptance_conditions, 143
 - compute_support_conditions, 143
 - compute_support_variables, 143
 - cycle_map, 142
 - format_state, 143
 - get_dict, 143
 - get_init_state, 143
 - neg_acceptance_conditions, 143
 - project_state, 144
 - state_is_accepting, 144
 - succ_iter, 144
 - support_conditions, 144
 - support_variables, 145
 - tgba_tba_proxy, 142
 - tgba_tba_proxy::operator=, 145
 - the_acceptance_cond_, 145
- Stack
 - yy::Stack, 78
- stack
 - spot::magic_search, 60
- stack.hh, 177
- stack_
 - yy::Slice, 77
- stack_type
 - spot::magic_search, 59
- start
 - spot::tgba_reachable_iterator, 125
 - spot::tgba_reachable_iterator_breadth_first, 128
 - spot::tgba_reachable_iterator_depth_first, 131
- state
 - spot::tgba_explicit, 110
- state.hh, 177
- state_
 - spot::state_bdd, 83
 - spot::state_explicit, 85
- state_bdd
 - spot::state_bdd, 82
- state_explicit
 - spot::state_explicit, 84
- state_is_accepting
 - spot::tgba_tba_proxy, 144
- state_iter_pair
 - spot::magic_search, 59
- state_name_map_
 - spot::tgba_explicit, 114
- state_product
 - spot::state_product, 86
- state_proposition
 - spot::emptiness_check, 49
- state_sequence
 - spot::emptiness_check, 49
- statebdd.hh, 178
- states
 - spot::emptiness_check::connected_component_set, 52

- step
 - spot::minato_isop::local_vars, 65
 - yy::Location, 57
- step_
 - spot::tgba_succ_iterator_product, 140
- succ_iter
 - spot::tgba, 93
 - spot::tgba_bdd_concrete, 98
 - spot::tgba_explicit, 113
 - spot::tgba_product, 121
 - spot::tgba_tba_proxy, 144
- succ_set_
 - spot::tgba_succ_iterator_concrete, 137
- succ_set_left_
 - spot::tgba_succ_iterator_concrete, 137
- succiter.hh, 179
- succiterconcrete.hh, 179
- suffix
 - spot::emptiness_check, 50
- super
 - spot::ltl::unabbreviate_logic_visitor, 146
 - spot::ltl::unabbreviate_ltl_visitor, 149
- support_conditions
 - spot::tgba, 93
 - spot::tgba_bdd_concrete, 99
 - spot::tgba_explicit, 113
 - spot::tgba_product, 121
 - spot::tgba_tba_proxy, 144
- support_variables
 - spot::tgba, 93
 - spot::tgba_bdd_concrete, 99
 - spot::tgba_explicit, 113
 - spot::tgba_product, 122
 - spot::tgba_tba_proxy, 145
- tgba
 - spot::tgba, 91
- tgba.hh, 180
- tgba_bdd_concrete
 - spot::tgba_bdd_concrete, 96
- tgba_bdd_concrete::operator=
 - spot::tgba_bdd_concrete, 99
- tgba_bdd_concrete_factory
 - spot::tgba_bdd_concrete_factory, 101
- tgba_bdd_core_data
 - spot::tgba_bdd_core_data, 104
- tgba_dupexp_bfs
 - spot, 16
- tgba_dupexp_dfs
 - spot, 16
- tgba_explicit
 - spot::tgba_explicit, 110
- tgba_explicit::operator=
 - spot::tgba_explicit, 113
- tgba_explicit_succ_iterator
 - spot::tgba_explicit_succ_iterator, 116
- tgba_parse
 - spot, 16
- tgba_parse_error
 - spot, 12
- tgba_parse_error_list
 - spot, 12
- tgba_product
 - spot::tgba_product, 119
- tgba_product::operator=
 - spot::tgba_product, 122
- tgba_reachable_iterator
 - spot::tgba_reachable_iterator, 124
- tgba_reachable_iterator_breadth_first
 - spot::tgba_reachable_iterator_breadth_first, 127
- tgba_reachable_iterator_depth_first
 - spot::tgba_reachable_iterator_depth_first, 130
- tgba_save_reachable
 - spot, 16
- tgba_succ_iterator_concrete
 - spot::tgba_succ_iterator_concrete, 135
- tgba_succ_iterator_product
 - spot::tgba_succ_iterator_product, 138
- tgba_tba_proxy
 - spot::tgba_tba_proxy, 142
- tgba_tba_proxy::operator=
 - spot::tgba_tba_proxy, 145
- tgbaabddconcrete.hh, 181
- tgbaabddconcretefactory.hh, 182
- tgbaabddconcreteproduct.hh, 182
- tgbaabddcoredata.hh, 183
- tgbaabddfactory.hh, 184
- tgbaexplicit.hh, 185
- tgbaproduct.hh, 185
- tgbatba.hh, 186
- the_acceptance_cond_
 - spot::tgba_tba_proxy, 145
- ThirdStep
 - spot::minato_isop::local_vars, 64
- to_string
 - spot::ltl, 20
- todo
 - spot::tgba_reachable_iterator_breadth_first, 128
 - spot::tgba_reachable_iterator_depth_first, 131
- todo_
 - spot::minato_isop, 63
- tostring.hh, 187
- True
 - spot::ltl::constant, 44

- true_instance
 - spot::ltl::constant, 45
- tstack
 - spot::magic_search, 60
- tstack_type
 - spot::magic_search, 59
- tunabbrev.hh, 187
- type
 - spot::ltl::binop, 38
 - spot::ltl::constant, 44
 - spot::ltl::multop, 67
 - spot::ltl::unop, 152
- U
 - spot::ltl::binop, 38
- unabbreviate_logic
 - spot::ltl, 20
- unabbreviate_logic_visitor
 - spot::ltl::unabbreviate_logic_visitor, 146
- unabbreviate_ltl
 - spot::ltl, 21
- unabbreviate_ltl_visitor
 - spot::ltl::unabbreviate_ltl_visitor, 149
- unop
 - spot::ltl::unop, 152
- unop.hh, 187
- unref
 - spot::ltl::atomic_prop, 25
 - spot::ltl::binop, 39
 - spot::ltl::constant, 45
 - spot::ltl::formula, 55
 - spot::ltl::multop, 69
 - spot::ltl::ref_formula, 76
 - spot::ltl::unop, 153
- unref_
 - spot::ltl::atomic_prop, 25
 - spot::ltl::binop, 40
 - spot::ltl::constant, 45
 - spot::ltl::formula, 55
 - spot::ltl::multop, 69
 - spot::ltl::ref_formula, 76
 - spot::ltl::unop, 153
- unregister_all_my_variables
 - spot::bdd_dict, 34
- v1
 - spot::minato_isop::local_vars, 65
- val
 - spot::ltl::constant, 46
- val_
 - spot::ltl::constant, 46
- val_name
 - spot::ltl::constant, 46
- var_formula_map
 - spot::bdd_dict, 35
- var_map
 - spot::bdd_dict, 35
- var_refs
 - spot::bdd_dict, 35
- var_set
 - spot::tgba_bdd_core_data, 106
- varandnext_set
 - spot::tgba_bdd_core_data, 107
- varnum
 - spot::bdd_allocator, 28
 - spot::bdd_dict, 35
- vars
 - spot::minato_isop::local_vars, 65
- vec
 - spot::ltl::multop, 67
- version
 - spot, 16
- version.hh, 188
- vf_map
 - spot::bdd_dict, 32
- visit
 - spot::ltl::clone_visitor, 41, 42
 - spot::ltl::const_visitor, 42, 43
 - spot::ltl::postfix_visitor, 73
 - spot::ltl::unabbreviate_logic_visitor, 147
 - spot::ltl::unabbreviate_ltl_visitor, 149
 - spot::ltl::visitor, 154
- visitor.hh, 188
- vr_map
 - spot::bdd_dict, 32
- X
 - spot::ltl::unop, 152
- x
 - spot::magic_search, 60
- Xor
 - spot::ltl::binop, 38
- yy, 21
 - operator+, 22
 - operator+=", 22
 - operator-, 22
 - operator=, 22
 - operator<=, 22
- yy::Location, 56
 - begin, 57
 - columns, 57
 - end, 57
 - lines, 57
 - Location, 57
 - step, 57
- yy::Position, 70
 - column, 71

- columns, [71](#)
- filename, [71](#)
- initial_column, [71](#)
- initial_line, [72](#)
- line, [72](#)
- lines, [71](#)
- Position, [71](#)
- yy::Slice, [77](#)
 - operator[], [77](#)
 - range_, [77](#)
 - Slice, [77](#)
 - stack_, [77](#)
- yy::Stack, [78](#)
 - begin, [79](#)
 - ConstIterator, [78](#)
 - end, [79](#)
 - height, [79](#)
 - Iterator, [78](#)
 - operator[], [79](#)
 - pop, [79](#)
 - push, [79](#)
 - seq_, [79](#)
 - Stack, [78](#)